TOUCHSPICE: PHYSICAL-VIRTUAL CIRCUIT EMULATOR

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science in Electrical Engineering

By

Kevin Christopher Peters

June 2012

COMMITTEE MEMBERSHIP

TITLE:	TOUCHSPICE: PHYSICAL-VIRTUAL CIRCUIT EMULATOR
AUTHOR:	Kevin Christopher Peters
DATE SUBMITTED:	June 2012
COMMITTEE CHAIR:	Dr. John Oliver, Assistant Professor
COMMITTEE MEMBER:	Dr. Lynne Slivovsky, Associate Professor
COMMITTEE MEMBER:	Dr. Bridget Benson, Assistant Professor

ABSTRACT

TOUCHSPICE: PHYSICAL-VIRTUAL CIRCUIT EMULATOR

Kevin Christopher Peters

This thesis involves the creation of a system of embedded touchscreen devices called touchSPICE to aid in the learning of basic circuits. Traditionally, circuit theory is taught to students in two different methods, lectures and laboratory exercises. Lectures focus on auditory and visual learning and are largely passive learning. Lab experiments allow students to physically interact with the circuits, and learn visually through viewing output waveforms from simulators or on measurement devices. The goal of the touchSPICE project is to develop a physical system for virtual, real-time SPICE simulation that mimics the laboratory experience. In touchSPICE, touchscreen devices act as circuit nodes that communicate with immediate neighbors using physical wires. Additionally, the nodes communicate wirelessly with a host computer, running a customized version of SPICE. Data is aggregated on the host computer and plotted in real-time. Changes in configuration of the nodes (component types and values), are then reflected on the host computer's display.

The efficacy of touchSPICE as a learning tool was evaluated by using anonymous surveys from 20 subjects including a pretest, followed by an interactive session with touchSPICE, and a follow-up posttest. Results collected showed that with a few changes to improve the responsiveness of the touchscreen, touchSPICE may be an effective method for teaching circuit theory. Additionally, users enjoyed the quick configuration time that touchSPICE provided, and felt that the real-time feedback of touchSPICE helped support understanding of how circuits operate.

Keywords: Touchscreen, SPICE, Bluetooth, Circuit Simulation, Electrical Engineering Education

ACKNOWLEDGMENTS

I would like to thank my parents and brother for all the help they have provided me along the way in school. Without them, I wouldn't be where I am today. I would also like to thank Josh O'Hara for co-developing the hardware used in this thesis with me. And last but not least, Dr. John Oliver for the conceptual idea of a touchscreen SPICE tool and funding the cost of the hardware.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

Every Electrical and Computer Engineering major coming through California Polytechnic State University, San Luis Obispo (Cal Poly) must learn the basics of circuit analysis. The primary method of learning is during lectures, which appeals to the auditory and visual learners, but many students still require the aesthetic aspect from the laboratory exercises. Currently, these labs rely on two techniques – SPICE simulations and breadboard analysis using measurement equipment.

Each of these methods has advantages and disadvantages in learning the circuit concepts. The method of breadboarding a circuit requires a lot more time and equipment. To change a resistance, the part must be replaced, or a decade box can be used. After you have the desired components, each part must be wired together, and more wires used for the measurement equipment. If something is configured incorrectly, including incorrect wiring, wrong values, or faulty components, the measurements will be incorrect, or even worse, a component or piece of equipment may blow out or break.[1]

SPICE provides a virtual environment for students to place any component at any value in any configuration without the risk of breaking any components. These simulations allow the users to plot out information relating to the voltage or current at any point in the circuit. The drawbacks of using SPICE to simulate circuits include a small learning curve (especially when using the netlist entry method), no option to simulate in real-time, and a severe disconnect from building a circuit. Through the previous experience comparing traditional circuit modeling and using SPICE, it is clear that the use of SPICE increases the ability for students to learn concepts. At the same

time, there are some limitations to the ease of use for students relating to the specifications of simulation parameters. When SPICE is applied to educational purposes, exercises are adjusted to accommodate for SPICE. To improve its effectiveness, a solution should be developed that evolves SPICE into a simple, user friendly tool with fast results.

1.1 SOLUTION

In order to improve the current methods for simulating circuits in lab, a new simulation tool was developed that includes the feel of building a circuit with the virtual nature of SPICE, while reducing the knowledge of SPICE and simulation parameters requirements. The solution proposed in this thesis involves a network of touchscreen devices to select components and build a circuit, which will then be simulated on a host computer running SPICE and updating information in at near real-time speed. This method for simulating circuits combines the efficiency and speed of SPICE with the physical feel of building a circuit creating a physical-virtual interface. Simulating a circuit requires the physical interaction for selecting components and wiring a circuit together, but uses virtual models for the simulation. Throughout my research, I was unable to find an example of SPICE being combined with touch screen devices. By creating this unique solution called touchSPICE, the amount of SPICE knowledge required has been reduced, and using a simple interface such as a touchscreen, the ability of a student to learn the concepts with ease should be increased. Through user evaluation involving two tests and an interactive session on touchSPICE, it can be shown that implementing touchSPICE to assist students in the learning of electrical engineering circuit concepts.

1.2 TOUCHSPICE SYSTEM OVERVIEW

The touchscreen SPICE simulator (touchSPICE) is built around multiple touchscreen devices and coordinated by a host computer. Each touchscreen device will represent a component selected by the user. The currently supported components include a resistor, capacitor, inductor, BJT, opamp, NMOS, diode, DC and AC voltage source, and a current source. All of these component screens, as well as the component selection screen can be seen in Appendix B. Most of the components will allow the user to select a value as well as a scale size from pico to mega. With components selected on the touchscreens, two wire cables can be attached to the connectors located on each device. Two of these connectors represent inputs to the component, and the other two are output connections. If a component has a single input or output, such as a resistor, any connections attached to the input connectors are connected during the host computer's method of building a netlist. Once the circuit is built, the devices are ready to communicate their information wirelessly to the host computer.

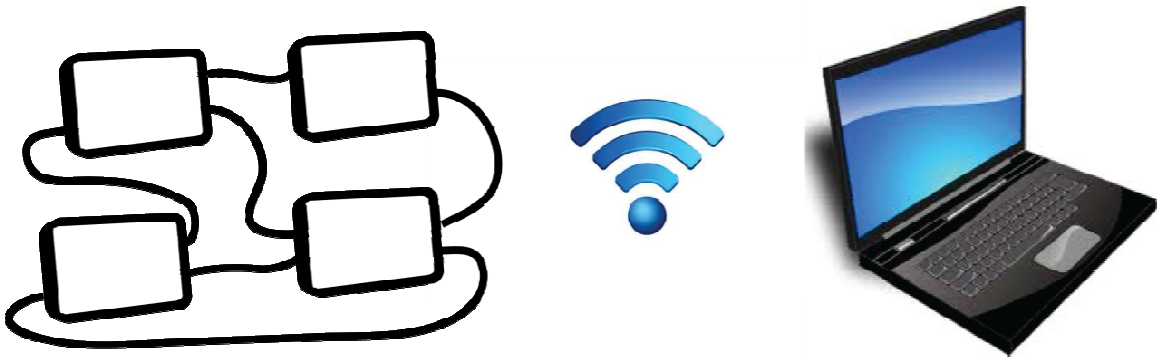


Figure 1: HARDWARE HIGH LEVEL OVERVIEW

Figure 1 shown above depicts a generic circuit communicating with the host computer. Each of the blank boxes on the left represents a separate circuit element. The

elements are wired together using a two-wire connection to communicate between nodes. Using a Bluetooth device, each node is able to communicate information relating to the components with the host computer.

The host computer's role begins with connecting to the wireless devices. Once the connections are established, the host will begin to poll the connected devices for component information as well as immediate neighbor listings. Neighbor information is coordinated by each touchscreen node. Eight of the available General Purpose Input/Output (GPIO) pins are used for internode communication. Each pin can be set as an input or output by setting different values in the GPIO registers. By setting four of the GPIO pins to inputs and four GPIO pins to outputs, they can be paired together to communicate with other nodes. The output pins can be set to a desired value, and the input pin on the other node will read the value set on the output pin it is attached to. When a node is asked for its neighbors, a process controlling the GPIOs is used to transfer two chars representing the node's name and a number representing the connector it is attached to. As updates to component specifications are made, the data being returned to the host computer will continue to update resulting in a near real-time simulation. The only updates that won't be detected are changes in wiring. This was left out of the update process due to the instability of removing wires during a simulation. Breaks in the circuit can result in errors during the SPICE simulation. If an update in wiring is required, the simulation must be stopped and then restarted when the updates are completed.

1.3 THESIS ORGANIZATION

The next section of this thesis will discuss the works related to this topic. Following the related work will be the complete system overview. With a grasp of the essential requirements of touchSPICE, the hardware component decisions will be broken down to explain why each component used in the final design was chosen. Following the hardware section, the node and host computer software processes will be described. To evaluate touchSPICE on its effectiveness as a circuit development tool, a pretest, interactive session, and posttest were given to participants in a user study. Each of these interactions will be described followed by an analysis section. The future of touchSPICE and the touchscreen development system will be discussed. All of the content in this thesis will then be wrapped up in the conclusion.

II. RELATED WORK

SPICE has already been used as a tool to aid in the teaching of basic circuit techniques. Laboratory courses have been developed to revolve around SPICE instead of standard physical component based experiments. These experimental courses have resulted in increased learning amongst students [7]. While using SPICE has been shown to increasing learning, an issue arises as the complexity of circuits begins to increase when simulating in SPICE. If a student doesn't understand the requirements needed to properly simulate a circuit using a transient response, time can be wasted during a trial and error process trying to find the right settings for the simulation [1].

Alternate simulation methods have been created to try and replace SPICE as a learning tool for circuit theory. SAPWIN was developed as a symbolic simulator for a faster and easier method of understanding some concepts [5]. Similar to touchSPICE, the goal in creating a new simulator interface was to reduce the amount of code needed to interface with the simulator. Using C++ to develop the program creates a highly modular program resulting in flexibility and the capability of reusing code. Both SPICE and SAPWIN lack the physical connection provided by touchSPICE, which we believe is an important part of the experience of building circuits.

TouchSPICE is similar to another platform called Sifteo. The Sifteo Cubes are 1.5 inch blocks with an LCD display that doubles as a button. These cubes can communicate with each other using near field communication [2]. The Sifteo Creativity Kit allows game development, which could be used to create a circuit simulator. The number of ways to interact with Sifteo Cubes poses a problem when considering a circuit simulator. Each cube has a single button and an accelerometer to measure motion. All of the

component selection and value changing would have to be implemented using a combination of those two features. The benefit to using Sifteo Cubes is the near-field communication. Placing two cubes near each other allows them to recognize their neighbor, and could be easily utilized to build a circuit.

Using a network of development boards to create a circuit introduces the possibility of using distributed processing. In order to implement distributed processing, a scheduling algorithm must be selected in order to allow the main tasks to complete while processing the background information. Using a shortest-time-first-maximum wait weighted-shortest-time-first (STFMW-WSTF) provides satisfactory results without causing excessive delays in the system units [4]. When the processes are time sensitive, they must be prioritized in order of which process must be completed first.

Parallel execution of processes is not new to SPICE simulations either. Some methods, such as parallel CPUs, GPU or FPGA offloading require modifying the SPICE kernel. Expression-level parallelism starts at the model description layer [8]. This method of parallelism detects subcircuits and assigns each subcircuit to an independent simulator. Because it is separated by subcircuits, simple circuits do not produce significant improvements in the simulation. Improving the analysis time does result in errors in measurement. At 52x analysis time improvement, the measurement error comes in at under 10%, and with a 17x time improvement, the error is reduced to less than 1%.

III. EVALUATION AND DESIGN OF THE TOUCHSPICE HARDWARE

The primary hardware components required for this interface include the touchscreen with a development board, wireless communication device, and inter-node communication. These three features will determine the ease of interfacing the user with the hardware and the capability of producing a real-time response of the simulator. **Figure 2** shows the hardware block diagram describing the nodes.

3.1 DEVELOPMENT BOARD EVALUATION

Deciding on the ideal development board could be narrowed down to form factor, ability to communicate through serial, local wire communication, processor speed and cost. The following development boards were considered as a possible platform to develop the SPICE interface.

3.1.1 FRIENDLYARM MINI2440/6410

The FriendlyARM Mini2440 shown in **Figure 3** and Mini6410 are complete development boards which utilize Samsung's S3C2440A 405MHz processor (ARM9) and Samsung's S3C6410 533MHz (ARM11) processor respectively.

Figure 2: FRIENDLY ARM MINI2440

FriendlyARM is an organization founded in Germany which now has over many global distributors and support worldwide. The Mini2440, being the smaller of the two, fits into a 4” square, while its larger model, the Mini6410, fits into a 4.3” square. The attached touch screen is a resistive touch device, but each board comes with a stylus to improve its interface. Both boards feature a GPIO with at least 30 pins allowing for data to be exchanged between boards and a serial port for use with the wireless interface. The price ranges from $120 for the small model and $190 for the larger model. Full specifications for the board can be seen in **Table 1**.

3.1.2 MICROBUILDER LPC1343

The MicroBuilder LPC1343 is a semi-complete development board which utilizes NXP's (Philips) LPC1343 72MHz ARM Cortex-M3 processor. The MicroBuilder was

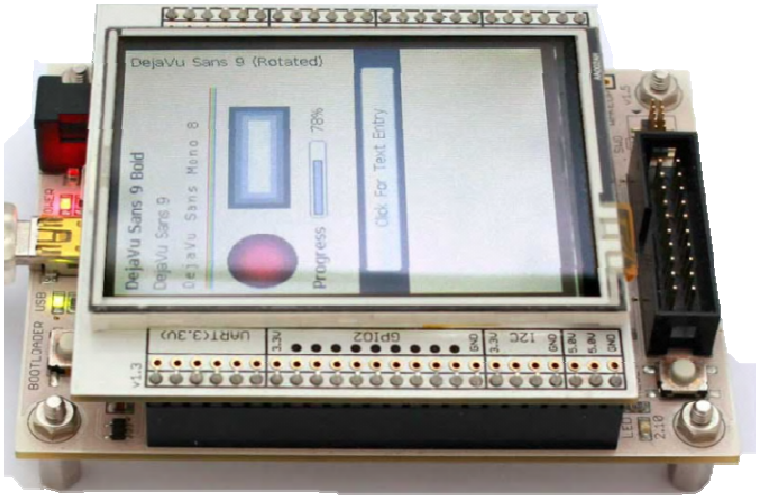


Figure 3: MICROBUILDER LPC1343

designed by an open-source organization and comes with a 2.8" thin film transistor touch screen. With up to 42 GPIO pins and a UART/I2C/SPI interface, it is capable of both exchanging information between boards and utilizing the wireless interface. The total cost of this option comes in at the lowest price of only $80. Full specifications can be seen in **Table 1.**

3.1.3 OMAP ZOOM EXPERIMENTER

The OMAP Zoom eXperimenter is a complete development board which utilizes TI's OMAP-LPC1343 C6-Integra DSP+ARM processor.

Figure 4: OMAP ZOOM EXPERIMENTER

This processor is a low-power applications processor based on an ARM926EJ-S and a C674x DSP core. The Zoom OMAP-L138 features a 4.3" LCD panel. This powerful development board features over 100 GPIO pins and a full serial interface to support local communication and wireless. With all of these features, the OMAP Zoom comes in as the most expensive board at $495. Full specifications can be seen in **Table 1.**

Table 1: DEVELOPMENT BOARD FULL SPECIFICATIONS

	Mini2440	Mini6410	MicroBuilder	OMAP Zoom
CPU/SOC	Samsung S3C2440A	Samsung S3C6410	NXP LPC1343	TI OMAP-L138
Core/Clock	ARM920T (405MHz)	ARM1176JZF-S (533 MHz)	ARM Cortex-M3 (72MHz)	ARM926EJ-S (456MHz)
RAM	64MB SDRAM	256MB DDR2	8KB SRAM	128MB DDR2
Flash	128MB NAND	1GB NAND	32KB On-Chip	N/A
LCD	Toppoly 3.5" 240x320	Sharp 4.3" 480x272	2.8" color TFT touchscreen	4.3" LCD Display
Touch-screen	Integrated 4 wire resistive touch screen interface	Integrated 4 wire resistive touch screen interface	Integrated 4 wire resistive touch screen interface	Integrated 4 wire resistive touch screen interface
Ethernet	RJ45 10/100M	RJ45 10/100M	N/A	RJ45 10/100M
Serial	1 DB9 RS232 COM0 2 TTL COM0, 1 with 4-wire sockets 1x4 2mm	1 DB9 RS232 COM0 4 TTL COM0, 1, 2, 3 with 4-wire sockets 1x4 2mm	UART/I2C/SPI	115.2kbps RS-232 debug serial port
USB	1 MiniUSB Device USB2.0 1 USB Host USB1.1	1 MiniUSB Device USB2.0 1 USB Host USB1.1	USB 2.0 HID and Mass Storage support built right into the ROM	1 USB SD card reader Serial cable (null-modem) 1 USB 2.0 high-speed On-the-Go interface 1 USB 1.1 full-speed host
Audio	Stereo in/out - 3.5mm Jack Built in Microphone	Stereo out - 3.5mm Jack Built in Microphone	N/A	Stereo in/out connectors TLV320AIC3106 audio codec
SD	SD Card standard size. Up to 32 GB	SD Card standard size. Up to 32 GB	SD Card standard	SD Card standard
SDIO	2 IIC channels 2 SPI channels	SDIO header for SDIO Wifi, etc. + SPI and IIC. 2x10 2mm	Full Speed USB, TTL UART, SPI and I2C interfaces	3 UART 2 I2C 2 SPI 1 SATA
User Inputs	6 push buttons 1 Pot 1 Power switch for 5V power	8 push buttons/interrupts 1 Reset push button 1 Pot 1 Power switch for 5V power 1 IR Receiver	1 Reset button 1 Bootload button	2 buttons
Indicators	4 User LEDs, Green 1 Power LED, Red	4 User LEDs, Green	1 User LED, Green	N/A

		1 Power LED, Red	1 Power LED, Red 1 USB LED, Green	
JTAG	10 Pin JTAG Header 2x5 2mm	10 Pin JTAG Header 2x5 2mm	Available to be soldered	Connectors for JTAG interface
LCD	40-pin header, 2x20 2mm	40-pin header, 2x20 2mm 41 pin Mini/Micro2550 style for FFC	Custom	Custom
GPIO	34-Pin header 2mm	30-pin header 2x15 2mm	Up to 42 General Purpose I/O (GPIO) pins with configurable pull-up/pull-down resistors	Up to 9 banks of 16-pins each
Timer/ Counters	N/A	N/A	4 general purpose counter/timers with 4 capture inputs and 13 match outputs Programmable WatchDog Timer	Three 64-Bit General-Purpose Timers
Camera	20 pin Camera Interface	CMOS CAM130 or similar. 2x10 2mm	N/A	N/A
Bus Expansion	N/A	2x20 2mm	20-pin expansion connector	N/A
External Int.	8-pin interrupt and user button connector	Buttons/Interrupts 1x10 2mm	N/A	Programmable
RTC	Battery Backed RTC	Battery Backed RTC	System tick timer for easy timekeeping	On-board, no battery backup
Software Support	Superboot - can auto-program flash from an SD card	Superboot - can auto-program flash from an SD card	No ARM or JTAG programmer is required! The chip comes with a built in USB bootloader that appears as a very small disk drive	U-Boot (bootloader/monitor)

OS	• Linux 2.6.x kernel supports all I/O with file system • Linux 2.6.32 with Qtopia 2.2.0 includes source and tools • WindowsCE Demo 5.0 (TS support) uC/OS-II	• Linux 2.6.x kernel supports all I/O with file system • Qtopia 2.2.0 + Qt + Qt/E • Android 2.0 • WindowsCE 6.0r3.NET uC/OS-II	• MicroBuilder has written a full software library • The software library includes complete GCC-based startup code and details on setting up an ARM development environment using open source tools. Linux 2.6.x	• Open source Linux DVSDK and demos • WinCE SDK • Code Composer Studio (CCS) v4 • DSP/BIOS v5 Board Support Library (BSL) sample programs
PCB/Size	6 layer 4”x4”	6 layer 4.3”x4.3”	3.75”x2.75”x0.5”	8”x6”x0.5”

3.1.4 DECISION

After reviewing the specified development boards, all of them met the requirements for being capable of producing wireless communication and local information exchange between nodes. **Table 2** shows the weighted point values that evaluated each choice. A low number represents and ideal choice.

Table 2: DEVELOPMENT BOARD DECISION MATRIX

	Mini2440	Mini6410	MicroBuilder	OMAP Zoom
Touchscreen Size (2)	2	1	4	2
Board Size (2)	2	3	1	4
Processor Speed	2	1	4	2
Cost (2)	2	3	1	4
Total	**14**	**15**	**16**	**22**

In order to allow students to use these devices on a lab bench, form factor became a very important specification. The OMAP Zoom eXperimenter came in as the largest board and the most expensive board, but contained the same size screen as the other devices. For these reasons, this board was eliminated as a final choice. The Mini2440 and Mini6410 have very similar specifications. While the Mini6410 is a better choice for screen size and processor speed, they are really increases beyond an already satisfactory level in the Mini2440. If it came to a decision between these two devices, there would be no need to spend the extra money for extra unnecessary processing power. The MicroBuilder screen was slightly too small for what will need to be displayed, but even more so, the processor was just much too slow for the response times needed to keep up with real-time updates. These conclusions resulted in the FriendlyARM Mini2440 being chosen as the ideal development board to deploy as the touchscreen nodes.

3.2 WIRELESS COMMUNICATION EVALUATION

The device chosen for wireless communication would be used as the transmission method between each individual node and the host computer. Two popular choices are Zigbee and Bluetooth. The key deciding factors between these two protocols are size and speed.

3.2.1 BLUETOOTH

Bluetooth operates on the 2.4GHz frequency band. The WT12-A UART Bluetooth device provides a simple and slim wireless device. This device operates on 3.3V and contains an internal antenna. Due to the popularity of Bluetooth devices, many computers have built in Bluetooth capabilities, and creating a simple interface on the host

computer. Specifications for the WT12-A list the range at 40m and a power level of 4 dBm and capable of 3000 kbps data rate. Being able to communicate quickly gives the computer more time for simulating and plotting data.

Figure 5: WT12-A BLUETOOTH MODULE

3.2.2 ZIGBEE

Zigbee modules provide a simple solution for creating a wireless network. A simple tool associated with setting up Zigbee devices makes it easy to configure a network of them. X-Bee devices employ the Zigbee protocol, but have an odd form factor. These modules are shaped like a pentagon, but require a breakout board, increasing the depth of the device. While some X-Bee devices are capable of ranges over 1km, they have a slower data transfer rate of only 250 kbps.

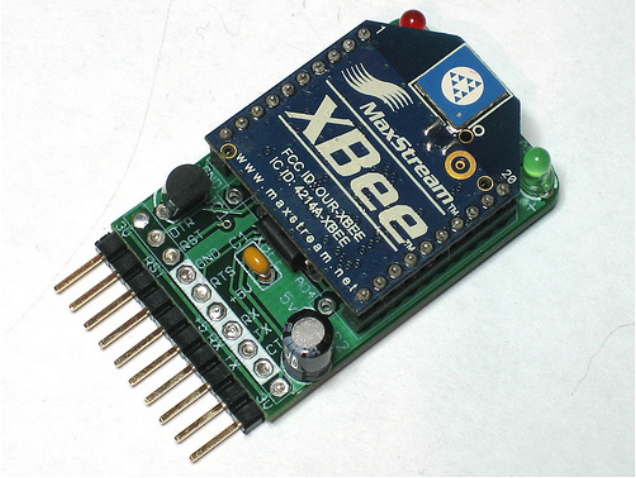


Figure 6: XBEE PRO ZIBGEE MODULE

3.2.3 DECISION

Bluetooth not only has a smaller form factor than Zigbee, but it also has a faster transfer rate. Zigbee is capable of a longer range for transmissions, but for this application, the computer will be close enough to the devices that range is not an issue. For these reasons, Bluetooth proves to be a stronger choice for wireless communication in this application. **Table 3** shows a comparison between the Bluetooth device and Zigbee module.

Table 3: WIRELESS MODULE COMPARISON

	Bluetooth	Zigbee
Frequency	2.4 GHz	2.4 GHz
Antenna	Internal	External/Internal
Data Rate	3000 kbps	250 kbps
Range	40 m	1 km
Power	4 dBm	10 dBm
Sensitivity	-84 dBm	-100 dBm
Size	14mm x 25.6mm x 2.4mm	27mm x 24mm x 9mm

Table 4: WIRELESS MODULE DECISION MATRIX

	Bluetooth	Zigbee
Data Rate (2)	1	2
Range	2	1
Power	2	1
Form Factor (2)	1	2
Total	**7**	**10**

3.3 INTERNODE COMMUNICATION EVALUATION

All of the development boards have plenty of extra GPIOs available for use. A "standard" connector needs to be determined so that the nodes can be connected together with wires to form a circuit. Since all blocks are to be physically identical, we must take into consideration the number of input and output connections required for each component. The majority of discrete components have 3 connections or less. By using four total connections, two on the left and two on the right side of the board, components such as transistors and opamps can be accommodated.

3.3.1 BANANA-TO-BANANA (1-WIRE) CONNECTION

Using banana-to-banana connections would utilize the current cables used in the senior project lab checkout window. The downside to this choice is the size of the connector socket. In order to use a single wire connection, the nodes would need to have GPIOs switch between input and output configurations. This would also require the nodes to sync up wirelessly so they know when to switch between the two setups.

3.3.2 DUAL JUMPER CABLE (2-WIRE) CONNECTION

Similar to the banana-to-banana cables, female jumper cables are also readily available. Having two wires instead of a single wire uses more GPIOs, but allows each wire to be a dedicated input or output. The jumper cables have a small socket and can be chained together to make a longer cable.

3.3.3 DECISION

The two choices for internode communication were weighed on availability connector size, durability, and ease of communication. Both connector size and ease of communication were weighted more heavily than the other fields. The dual jumper setup offers a smaller interface with increased communication by using dedicated input and output GPIOs. **Table 5** shown below shows the weighted evaluation of both wiring methods, resulting in choosing the 2-wire method.

Table 5: INTERNODE COMMINICATION DECISION MATRIX

	Banana-to-banana	Dual Jumper
Availability	1	2
Connector Size (2)	2	1
Durability	1	2
Ease of Communication (2)	2	1
Total	**10**	**8**

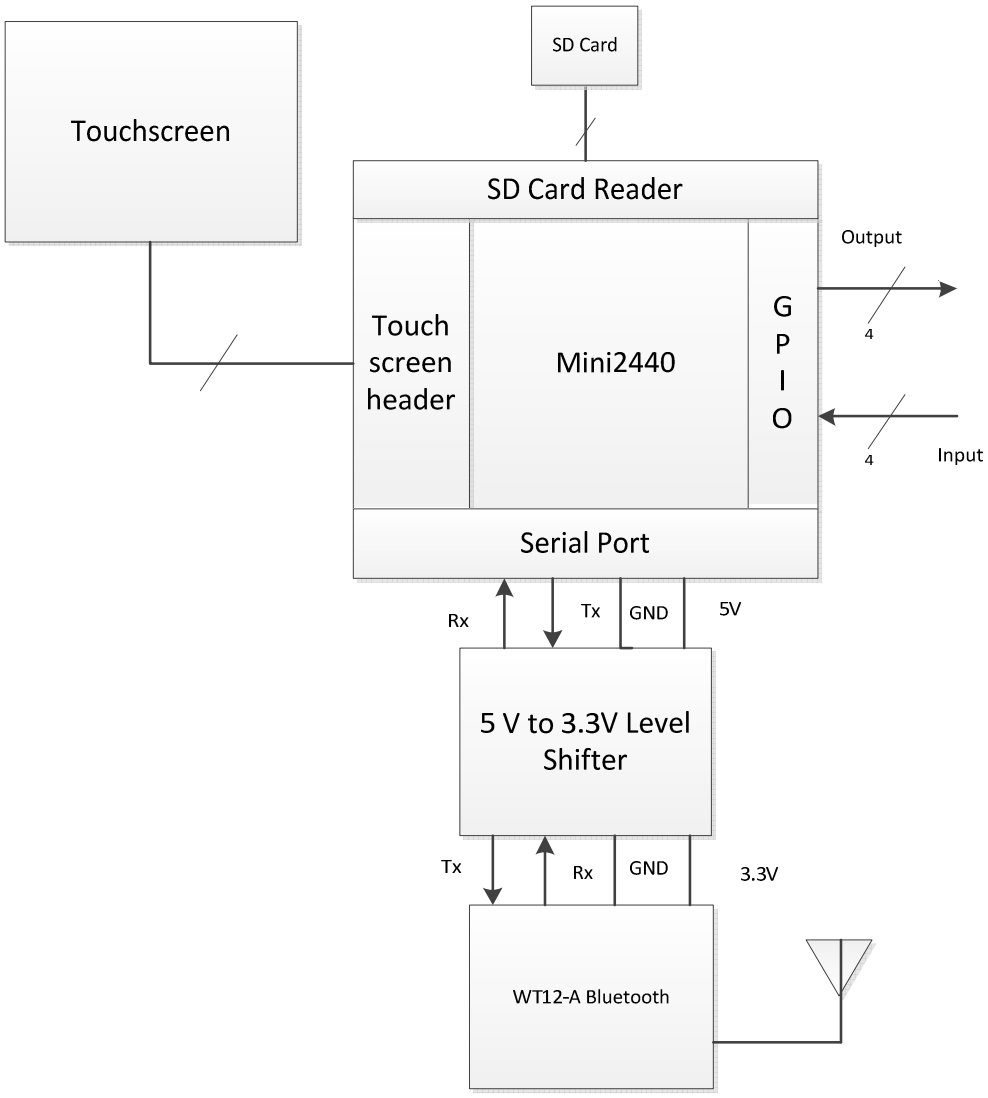


Figure 7: HARDWARE BLOCK DIAGRAM

Figure 7 shows the resulting hardware block diagram using the chosen hardware components. The Mini2440's touchscreen header is used to interface with the attached touchscreen. Programs are loaded onto the device using the SD Card reader built into the board. Internode communication utilizes eight GPIOs to create four input and four output connections. Finally, the serial port is utilized to communicate with the WT12-A Bluetooth device after passing through a 5V to 3.3V level shifter.

IV. TOUCHSPICE SOFTWARE DESIGN

The software developed for both the host computer and the touchscreen nodes was developed in a Linux Mint environment using Nokia's Qt Creator. Qt Creator has a built in GUI developer and a C++ compiler. Linux Mint is based off of the Debian distribution and is free to use. In order to produce real-time simulations, the data needs to be coordinated quickly and simulated only when necessary. **Figure 8** shows the high level software between the host computer and the touchscreen nodes.

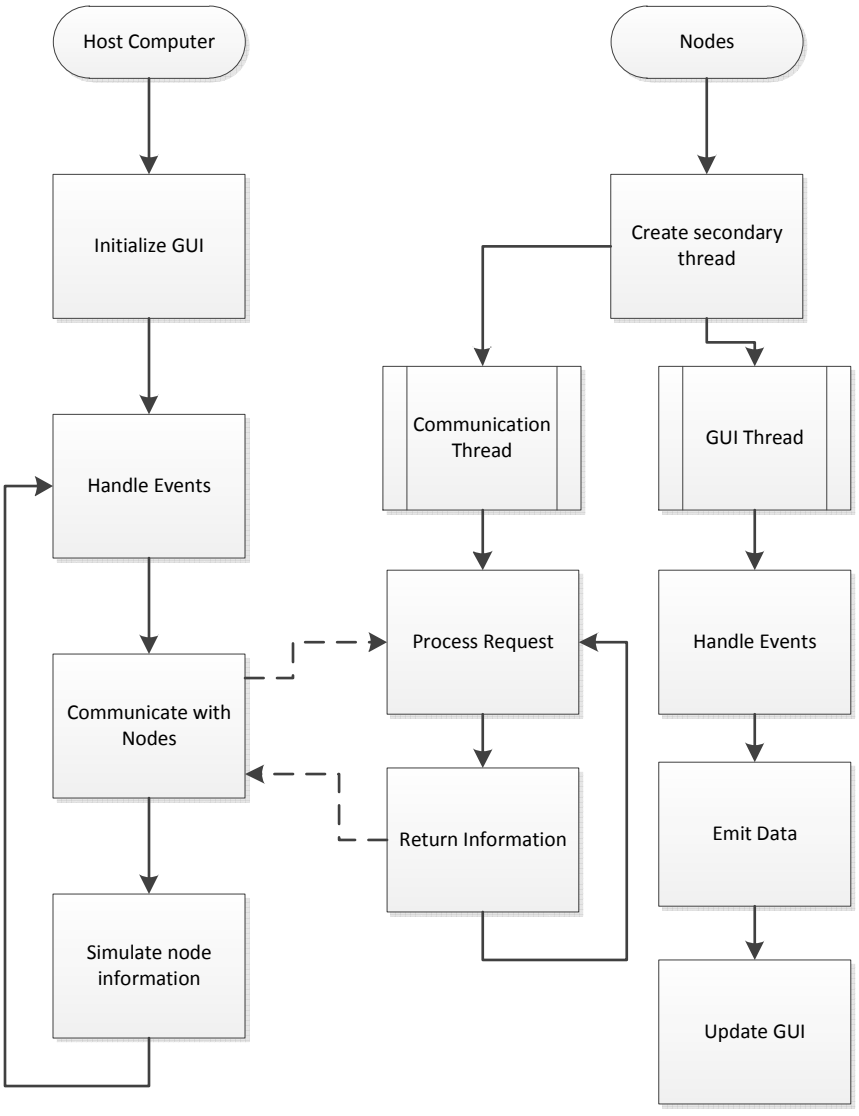


Figure 8: HIGH LEVEL SOFTWARE FLOW DIAGRAM

The host computer begins by initializing the user interface and establishes the event handler. When a simulation event is encountered, it enters the loop that communicates with the nodes. The dashed lines show the wireless communication between the host computer and the nodes. On the node's end, two threads are created; one thread handles the GUI interface and the other thread handles communication requests.

4.1 NODE SOFTWARE

The primary functions of the node software are to update the display and respond to information requests received from the host computer. In order to accomplish both functions in real-time, two threads are created. The primary thread contains the event handler, which processes actions made by the user on the touchscreen. The secondary thread manages communication with the host computer and internode communication. All of the software flow diagrams are shown in **Appendix C.** After an update is made on the touchscreen, the primary thread emits the value to the secondary thread. Meanwhile, the secondary thread constantly reads its wireless port's input buffer waiting for an instruction from the host computer. The node software will process the instruction and send back the requested data. If the request is for neighbor information, the secondary thread enters a sub-process for exchanging information over GPIO.

Neighbor identification begins with the node that received the command to find its neighbors by changing its first output GPIO pin to a '1'. The neighboring node that received the '1' responds by mimicking the '1'. The initiating node proceeds to send a '0' and then another '1' to let the receiving node that data is coming. After the data transfer setup is complete, 12 bits of data are transferred. The first eight bits represent the data

and the next four bits are parity bits to ensure the correct data was received. If an error in the parity was read, the same node will attempt to resend the data. Upon a successful transfer, the receiving node becomes the sender and transmits its own information. The first data transfer from each node is the eight bit code representing the node name, which is in the ASCII range ‘A’-‘G’. The second transfer from each node is the eight bit code for the connection number represented by and ASCII ‘0’-‘3’.

4.2 HOST SOFTWARE

The user interface on the host computer is minimal, but is used as the main coordinator in the system. **Figure 9** shows the software flow diagram of the host computer. Upon running the application, the program will setup the user interface and disconnect any current Bluetooth connections. In order to start simulating circuits, the user must select the “connect” button first to open the Bluetooth connections. After the connections are made, the “simulate” button may be used to begin the simulating process. The program will continue to update information and simulate until the “stop” button is clicked.

During the simulation process, the host computer coordinates communication and uses that information to build a SPICE netlist, simulate it, and plot it. The process begins by asking every node what other nodes it is connected to. After the neighbor information is collected, an infinite loop is entered and the only escape is using the stop button. The infinite loop begins by asking each connected node what component it is, followed by its value, and options for measurement. Each time new information is collected, the new data is compared to the previously stored data to check for changes. Only if a change is detected will the program generate a netlist, simulate it and create a new plot.

To generate a netlist, two main steps are required. The first step is decoding the neighbor information collected at the beginning of the simulation to build the wires. This process is discussed in a future section. The second part is simply writing the touchscreen node information into the netlist file. Once generated, a simple batch simulation is run using the ngspice SPICE simulator. After the simulation is complete, gnuplot uses the information stored in the output file to plot the specified voltages and currents.

The time it takes to execute these tasks is very minimal and updates can be seen in almost real-time. Action events, such as clicking buttons, are handled by the event handler. Since this is only a single thread process, only one button can be recognized at a time until the previous selection has been completed. This functionality results in a small delay between clicking the “stop” button and ending the current simulation. If multiple buttons are clicked in a row, they will be queued up until the previous process is complete.

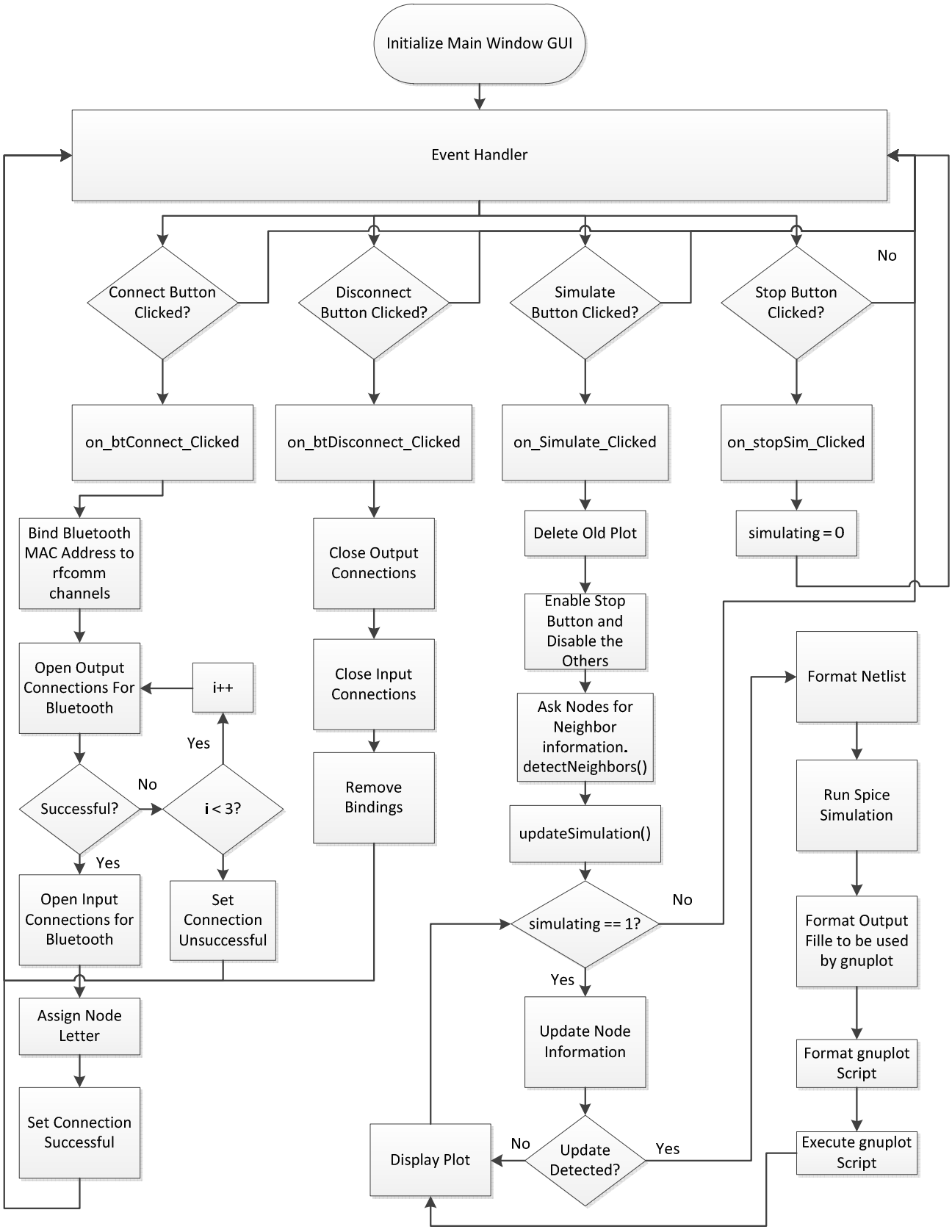


Figure 9: HOST COMPUTER SOFTWARE FLOW DIAGRAM

4.2.1 DISTRIBUTED VS. CENTRALIZED PROCESSING

Simulating the netlist can either be executed on the host computer, or distributed amongst the nodes. This stage is a time sensitive stage as it will determine whether the updates produced from changing components can be shown in real-time or not. Processing on a single node would not be a viable option, unless it was a dedicated simulation node. Attempting to run the full simulation on a node that is also representing a component in the circuit would severely diminish its capability of making real-time updates. Similarly, distributing the simulation amongst the active nodes would result in negative results. While the active functions may be given priority time on the processor, the simulation would be pushed back, delaying the output of the simulation. Each of these methods would introduce an unwanted delay, resulting in a poor response time for real-time updates.

Centralized processing would take place entirely on the host computer. This is the best option for creating a real-time response in the circuit updates. By allowing the processors on the individual nodes to handle updates, user response time is improved. Another benefit to performing simulations on the computer is the increased processor speed. Each node has a 405 MHz processor, but the computer used in testing has a 2.67 GHz quad core processor. Maintaining the simulations on the host computer also gives the nodes more time to handle updates during the simulation time. Problems were generated during the user study when the host computer continually polled for information while no updates were made. These requests slowed down the nodes to a point where making a change was difficult, but once a single modification was made, the

host computer had a longer delay to run the simulation and process the data, allowing more time for the nodes to process user requests.

In order to make distributed processing a feasible solution for processing the simulations, the processor speed on the nodes must be increased. Increasing the processor speed would allow the nodes to handle user interaction on the touchscreen and simulate the netlist using SPICE. And ideal solution would use a scheduling algorithm that would prioritize the simulation over the normal node processes in order to produce the real-time output. Distributed processing would become a more ideal solution as circuit complexity increases due to the increased probability of separate subcircuits contained in the overall circuit.

4.3 WIRELESS COMMUNICATION SOFTWARE

One of the key aspects to using a network of touchscreen devices for building a SPICE simulation is communicating data to the host computer. There are four aspects to communication involved in this process – connect, send, receive and disconnect.

Opening connections is done in three parts. The first step is to bind the specific MAC Addresses to rfcomm ports. Using a simple bind call, the first time communication is attempted on an rfcomm port with a binding; it will direct that communication to the specified MAC Address. [3] Step two attempts to open an output channel to each rfcomm with a binding. To open an output connection, or stream, the C++ class ofstream is used. Ofstream can be used to open a stream to files or out ports, such as rfcomm. If the output connection is successful, the final step opens and input channel so information can be received from the nodes. Similar to ofstream, the input stream is opened using ifstream.

With both the ifstream and ofstream opened on an rfcomm port, communication is fully established to a node.

Sending and receiving are closely tied together. Every message sent from the host computer knows a response is coming. Using the ‘write’ function associated with the ofstream class, a string can be sent over Bluetooth. Each node waits for messages to be received and reads the buffer one char at a time. These chars read are codes telling the node to send certain information. Information sent back to the host computer is organized into a string followed by a ‘!’. Back on the host computer, ifstream’s ‘readsome’ function reads a char at a time until the ‘!’ is read. If the ‘!’ is not read within an allotted time frame, the loop reading in characters returns a timeout. **Figure 10** shows the flowchart of the communication protocol on the host computer’s end. Temporary variables are used to read the input buffer. All data transfers begin with the host computer sending a char to a node indicating the data it wants to receive. The process then enters a loop while reading data. Each read attempts to read a single char out of the input buffer. If a NULL is read, the process attempts to read another char from the buffer. If a valid char is read, it is stored in another temporary variable. This process continues until a ‘!’ is received. Upon reading a ‘!’, the temporary variable holding the valid chars is returned in the form of a string.

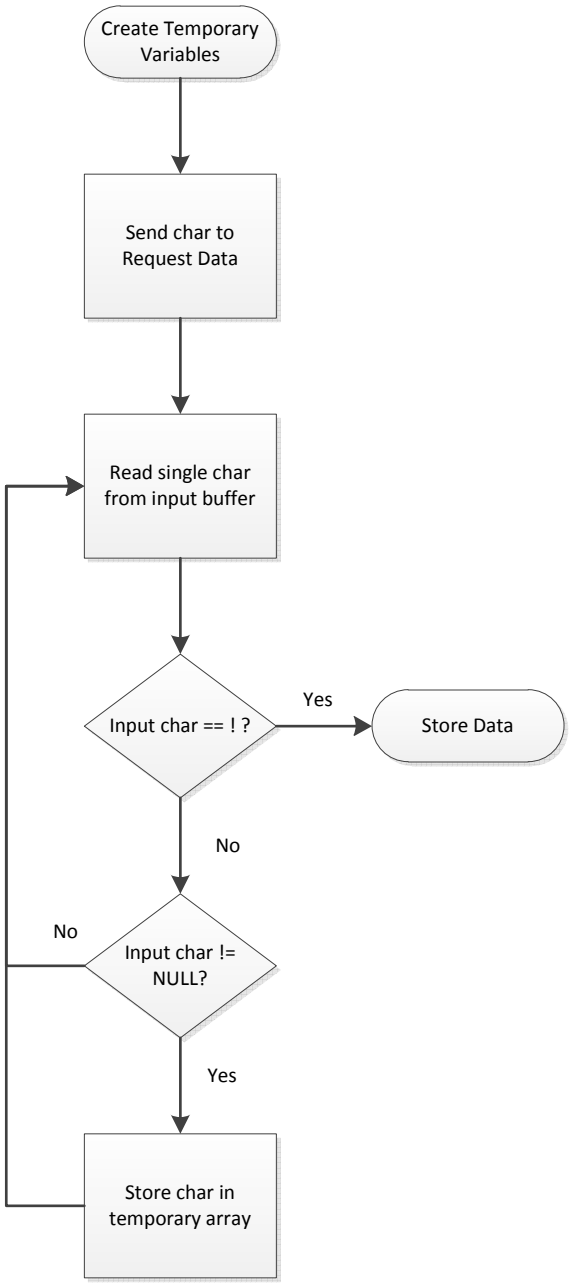


Figure 10: HOST COMPUTER COMMUNICATION FLOW DIAGRAM

Performing a disconnect is very similar to the connect process. The ifstream and ofstream connections are closed first. Then the rfcomm bindings are removed to clear up connections in the future.

4.4 NETLIST WIRE NUMBERING

Creating the netlist wire numbers is the last big step in building the netlist. Unfortunately, these wire names can't be created using the neighbor information codes, but must be decoded first. Each code has two chars representing its neighbor, the first char denotes the node name, and the second char denotes its port. Ports 0 and 1 represent the two inputs and ports 2 and 3 represent the outputs. Due to the possibility that a component can have a single input such as a resistor, or two inputs like an opamp, each component must be checked to see if the input ports are actually tied together.

Starting the numbering process requires a voltage source to find the ground. Once the ground has been numbered, the algorithm proceeds node by node assigning numbers to the outputs one at a time. If the outputs happen to be tied together (resistor, capacitor, etc.), they are given the same number. As each output is numbered, a sub-process checks the other nodes to see if they possess the neighbor code of the output that was just assigned a number. Once all the outputs are assigned a number, the process is repeated for the input nodes. If a node has already been assigned a number, the sub-process that checks the neighbor codes is given the already assigned value to prevent a node from getting its inputs or outputs assigned multiple numbers.

V. USER STUDY

In order to test the educational benefits of touchSPICE, a user study was conducted. The user base included current students at Cal Poly ranging from freshman to graduate students. To protect any personal information relating to the students participating in the study, no identification information such as name, email or phone number were collected. Each participant was assigned a user number so results of the pretest and posttest could be tracked to show improvements in each subject's results. Users filled out a survey before using touchSPICE and another survey after interacting with the setup. Instead of printing out forms for every user who participated in the study, a Google Survey Form was created and laptops were setup for the users to take the surveys. Information collected by the Google Survey is automatically entered into an Excel Spreadsheet for easy data collection. The forms used in the user study are shown in **Appendix D**.

5.1 PRETEST

The pretest was designed to gauge each user's knowledge of a few circuit concepts, as well as collect class level standing and circuit testing experience. 20 students participated in the pretest with a majority of the students being junior standing. **Table 6** shows the breakdown of students who participated in the survey. Of these 20 students, 19

Table 6: PRETEST GRADE LEVEL BREAKDOWN

Grade Level	Freshman	Sophomore	Junior	Senior	Graduate
Number of Participants	4	2	11	2	1

of them had used a SPICE simulator, and all 19 of those students had used LTspice IV. This helped establish a good basis for comparing touchSPICE with a commonly used SPICE simulator. To evaluate the subjects' current knowledge of some circuit concepts,

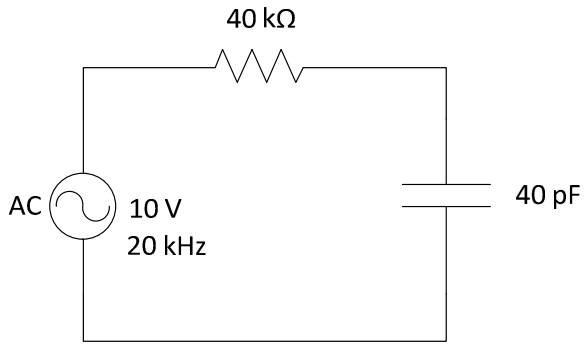


Figure 11: RC LOW-PASS FILTER CIRCUIT

four circuits were designed. The first circuit shown in **Figure 11** depicts a RC low-pass filter. Students were asked how increasing the value of the resistor would affect the cutoff frequency. Of the 20 students participating, only seven students knew the correct answer. The remainder of the students split between a wrong answer and replying with 'don't know'. Replying with 'don't know' was used as a response for students who hadn't learned the concept yet or didn't remember how to concept worked. A full breakdown of the how many students answered the questions right or wrong can be seen in **Table 7.**

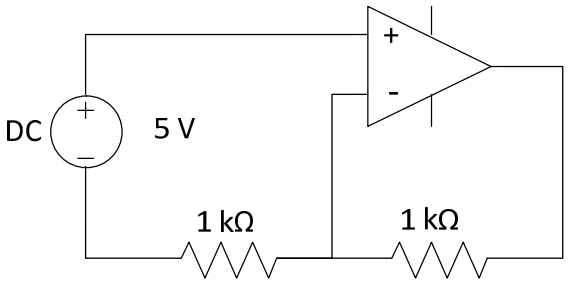


Figure 12: NON-INVERTING OPAMP CIRCUIT

The second circuit was a non-inverting opamp as shown in **Figure 12**. Students were asked whether the gain would increase or decrease as the feedback resistor's value was increased. A majority of the students, 14, were able to answer this question correctly, while only four got it wrong and two didn't know. The third circuit shown in **Figure 13**

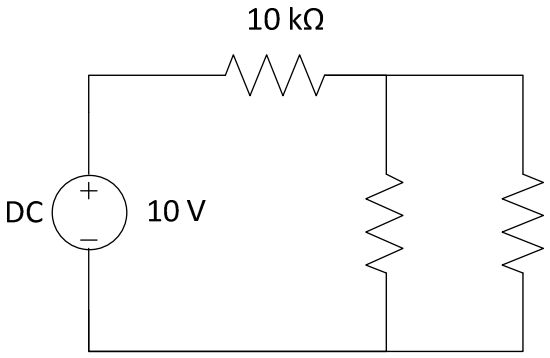


Figure 13: PARALLEL RESISTOR CIRCUIT

was a DC voltage source in series with one resistor, and two resistors in parallel. This circuit tested the concept of current in parallel resistors. If one of the parallel resistors had its value decreased, how is the current through the parallel branches affected? Out of all the questions on the pretest, this was the most commonly understood question. 17 students got the correct answer, only 3 got it wrong, and no students replied with 'don't know'. The final circuit shown in **Figure 14** was a resistor in series with a diode. Students were asked to calculate the value of the resistor that would result in a 20 mA current through the diode.

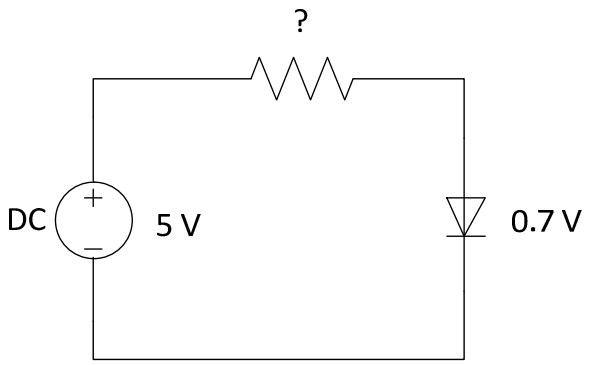


Figure 14: DIODE CIRCUIT

Half of the students got the question correct, 9 got it wrong, and one answered 'don't know'. Most of the students who answered this question incorrectly ignored the voltage drop in the resistor. A majority of the students assumed a 0.7 V drop through the diode and a few asked what the drop across the diode was.

Table 7: PRETEST STUDENT ANSWER RESULTS

	Student's Response (number of students)		
Circuit	Right	Wrong	Don't Know
Low-pass Filter	7	6	7
Opamp	14	4	2
Parallel Resistors	17	3	0
Diode	10	9	1
Total	**48**	**22**	**10**

5.2 USER INTERACTION

After completing the pretest, the students were directed towards the touchSPICE setup. Only four nodes were activated during the user study due to the complexity of the circuits being tested. Because the touchSPICE interface was also being tested as a part of a human computer interaction (HCI) study, users were given no instructions on how to use touchSPICE. Students were asked to simulate as many of the circuits from the pretest

that they wanted to so they could correctly answer any of the questions that were previously asked. While most students used their time on touchSPICE to simulate the pretest questions to find the correct answer to the questions, some of the students testing the system decided to build other circuits. If the student hadn't used a SPICE simulator recently, they were asked to use LTspice IV to simulate a circuit or two to refresh their memory on simulating circuits via SPICE. While watching some of the students on LTspice IV, it was apparent where students were struggling. They tended to be quick to build the circuit, but had trouble simulating it. This result demonstrated the struggles of students expressed in the introduction and seen in previous labs that introduced SPICE as a learning method. The following series of figures depicts the building and simulation of the low-pass filter.

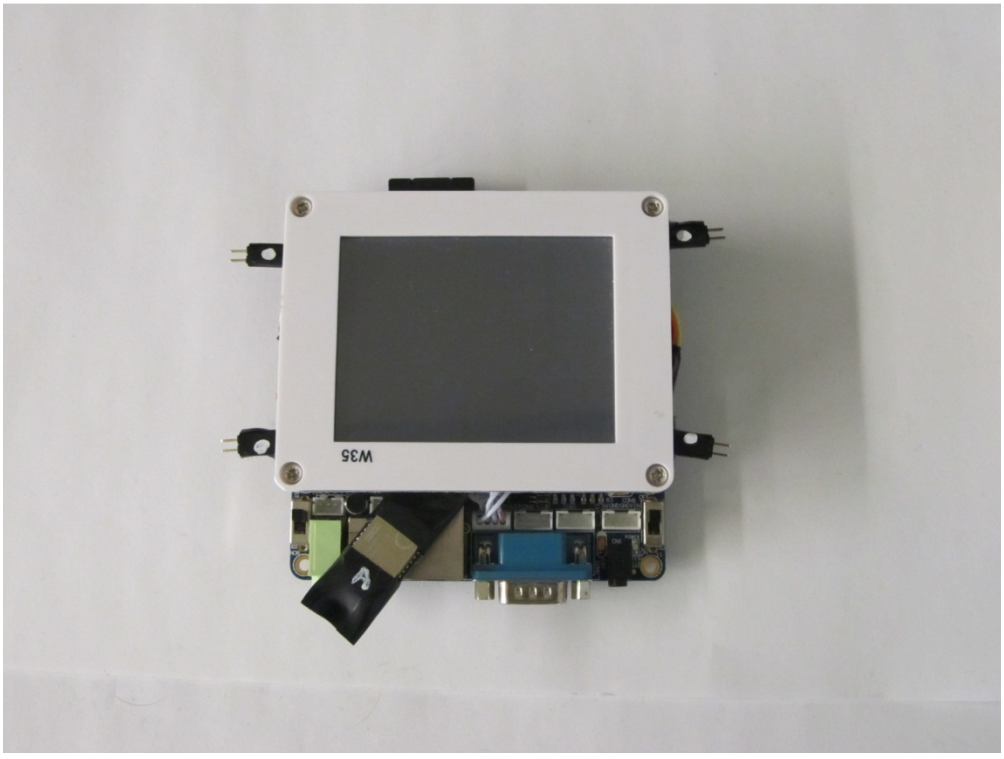


Figure 15: INDIVIDUAL NODE

Figure 15 shows a single node without power. At the top is the SD card containing the touchSPICE software. The four connectors on the corners are used for internode communication. Finally, the Bluetooth module is sticking out of the bottom of the screen.

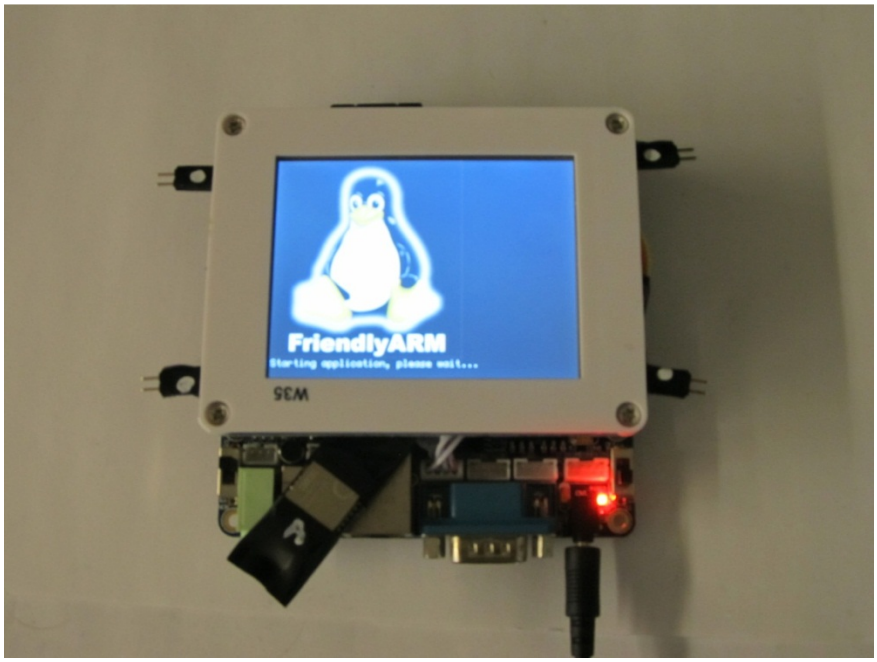


Figure 16: NODE BOOT SCREEN

Figure 16 displays a singular node during boot up. The Linux penguin is displayed while the program is started up.

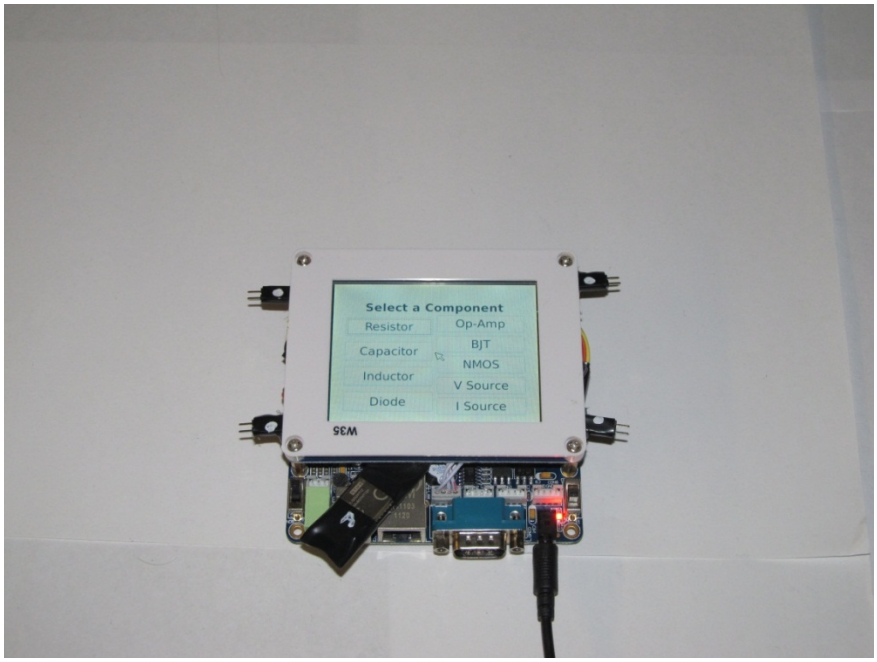


Figure 17: NODE AT COMPONENT SELECTION SCREEN

Figure 17 displays the node after it has finished booting into the touchSPICE program. The initial screen displayed is the component selection screen.

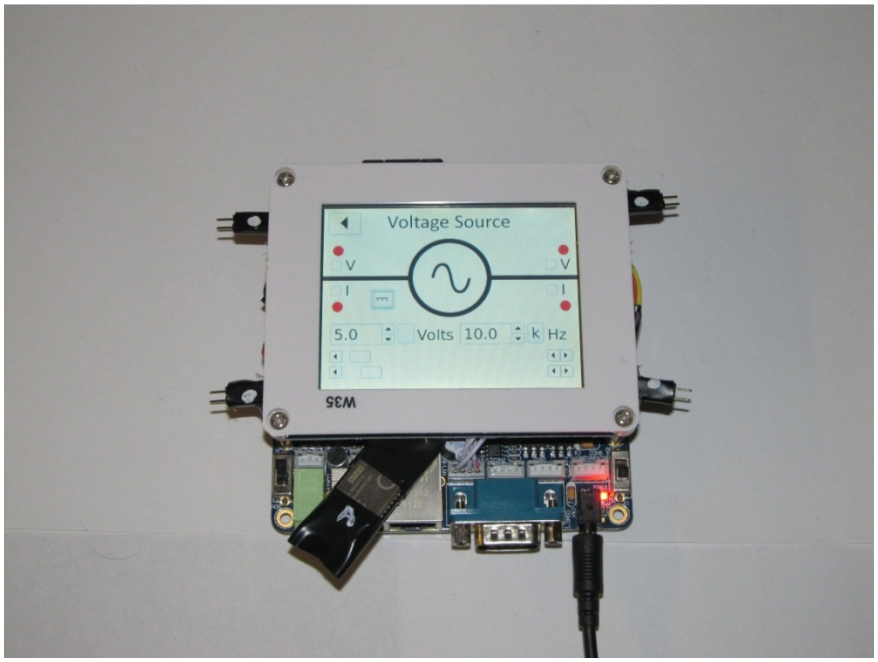


Figure 18: AC VOLTAGE SOURCE

Figure 18 displays a node after a component has been selected. The current component is a voltage source set to AC mode. No connections have been made.

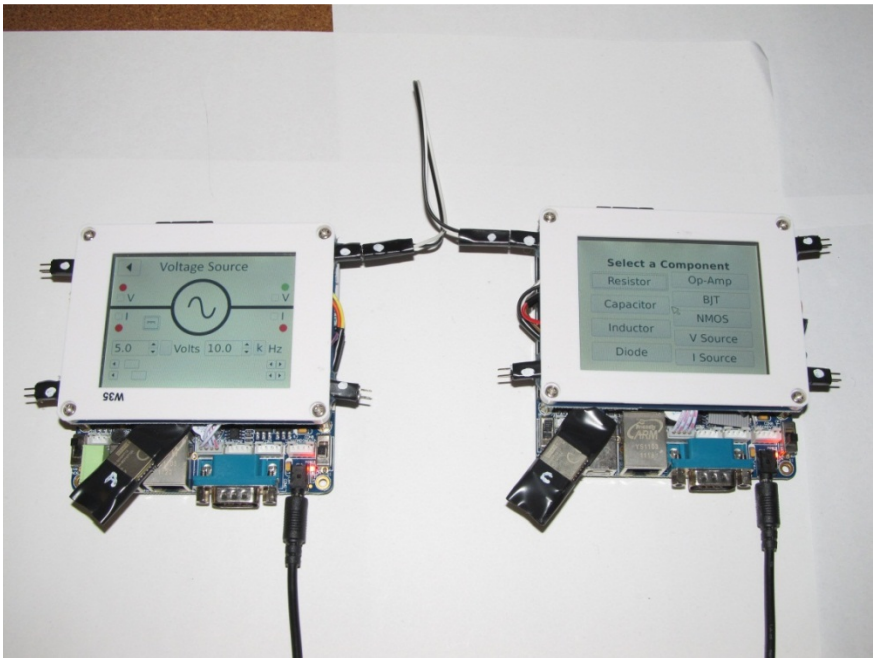


Figure 19: TWO NODES CONNECTED TOGETHER

Figure 19 shows a second node added to the network. The wires are connected with the white dots oriented in the same direction.

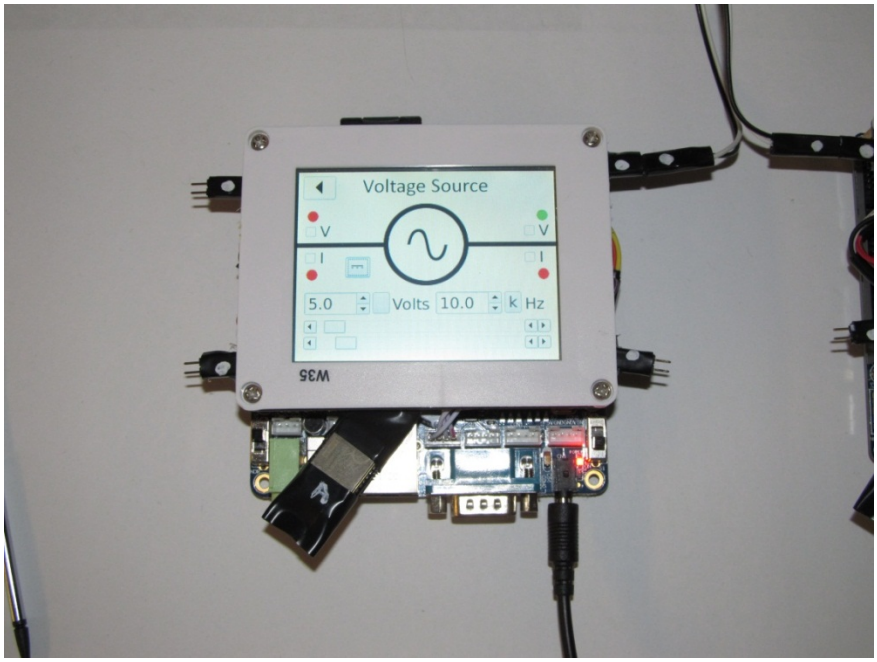


Figure 20: SUCCESSFULLY CONNECTED NODE

Figure 20 shows the first node after the connection was made. The dot indicating the connection status on the top right has turned green to indicate a good connection.

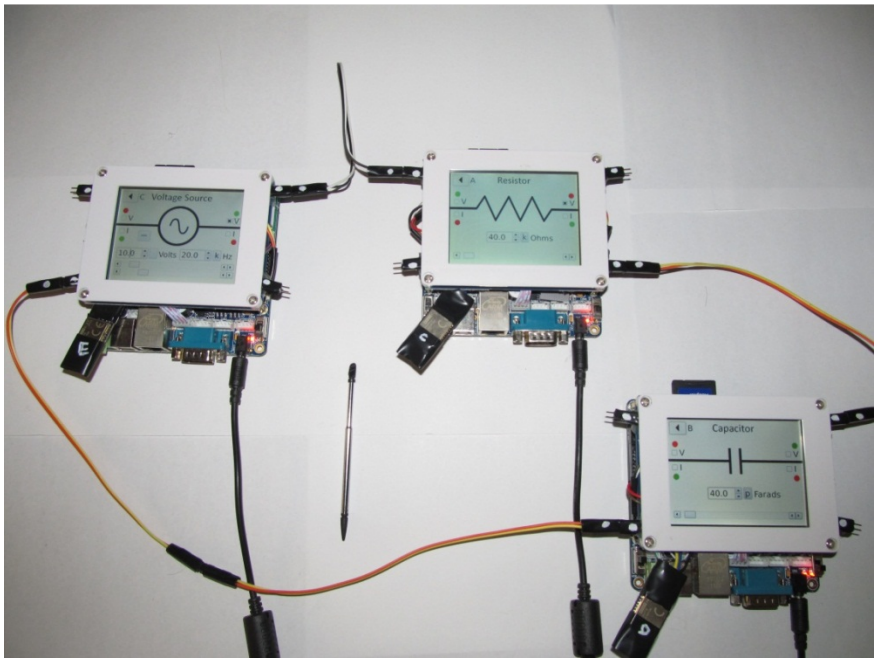


Figure 21: COMPLETED LOW-PASS FILTER

Figure 21 shows a successfully built low-pass filter. Each dot with a wire connected to it has turned green to signify good connections. The wire connecting the capacitor node to the voltage node has been extended by adding a jumper between two of the standard connectors.

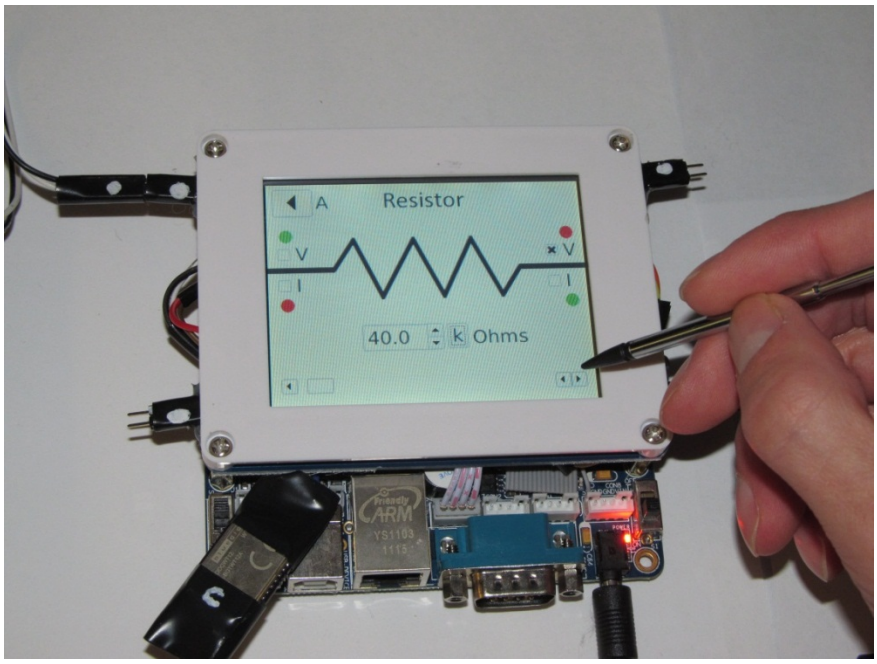


Figure 22: ADJUSTMENT VIA SCROLL ARROWS

Figure 22 shows one method of changing the value of the component. Using the arrows on the right or left side of the scroll bar will increment or decrement the value in the box by 1.

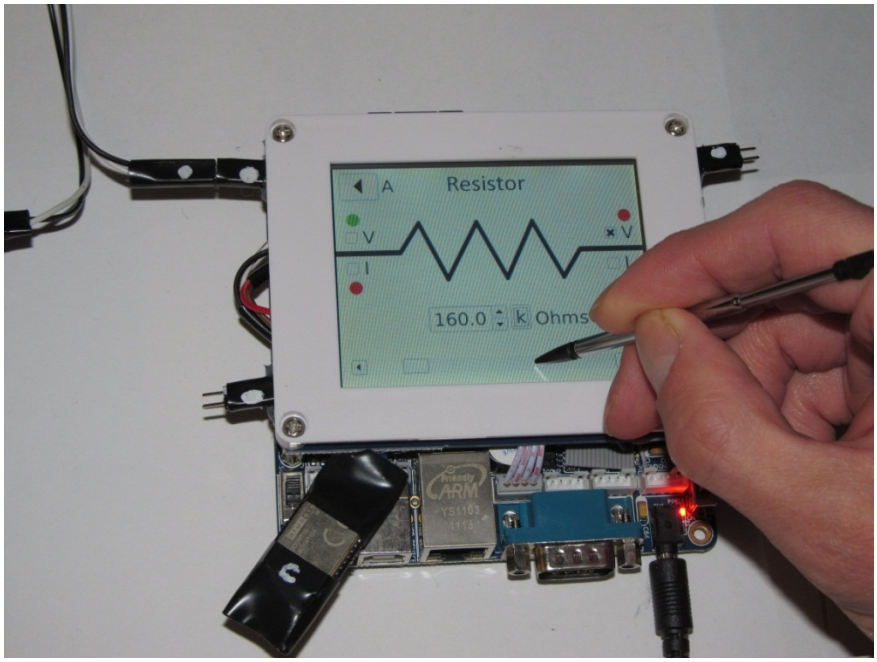


Figure 23: ADJUSTMENT VIA TAPPING SCROLL BAR

Figure 23 shows an alternate method of adjusting the component value. Tapping on the scroll bar away from the slider will adjust the value by 10 at a time.

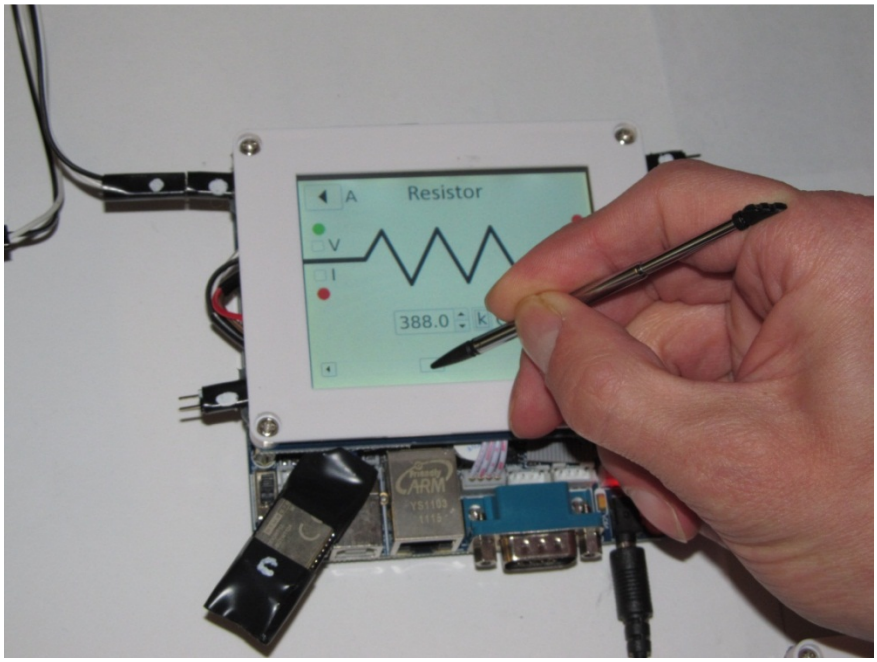


Figure 24: ADJUSTMENT VIA DRAGGING THE SCROLL BAR

Figure 24 shows the fastest way of changing a component's value. Dragging the slider on the scroll bar will result in fast changes.

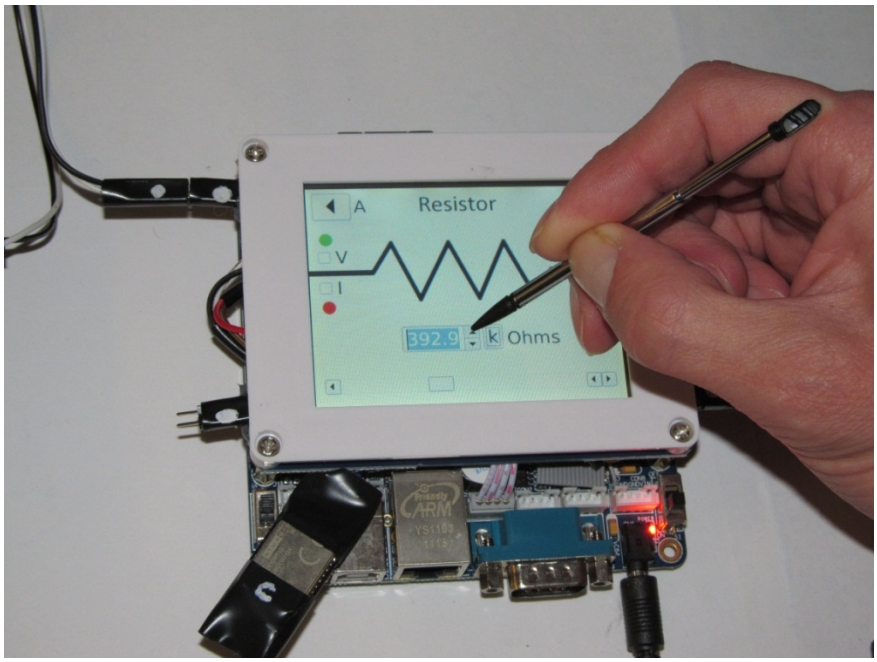


Figure 25: MINOR ADJUSTMENT VIA SPIN BOX ARROWS

Figure 25 shows how to change a component value in finer increments. Using the up arrow will add 0.1 to the value of the component. The down arrow will decrease by 0.1 unless the value is already a whole number. When it's a whole number, it will decrement by 1.

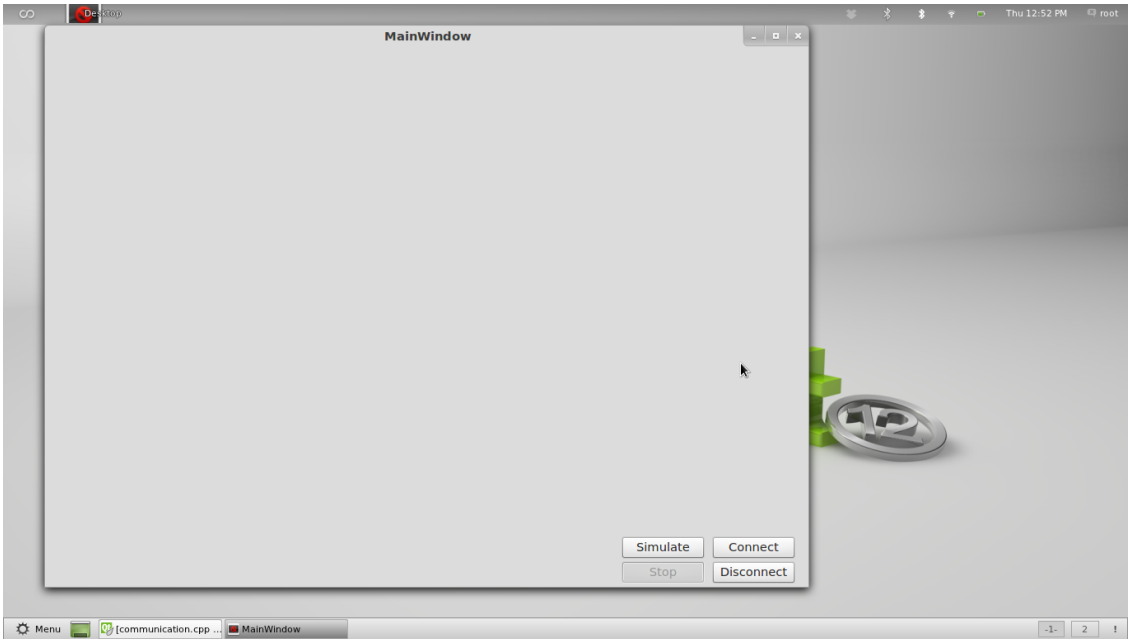


Figure 26: HOST COMPUTER INTERFACE STARTUP

Figure 26 is displaying the GUI on the host computer when it is first started. All of the buttons are active except for the stop button.

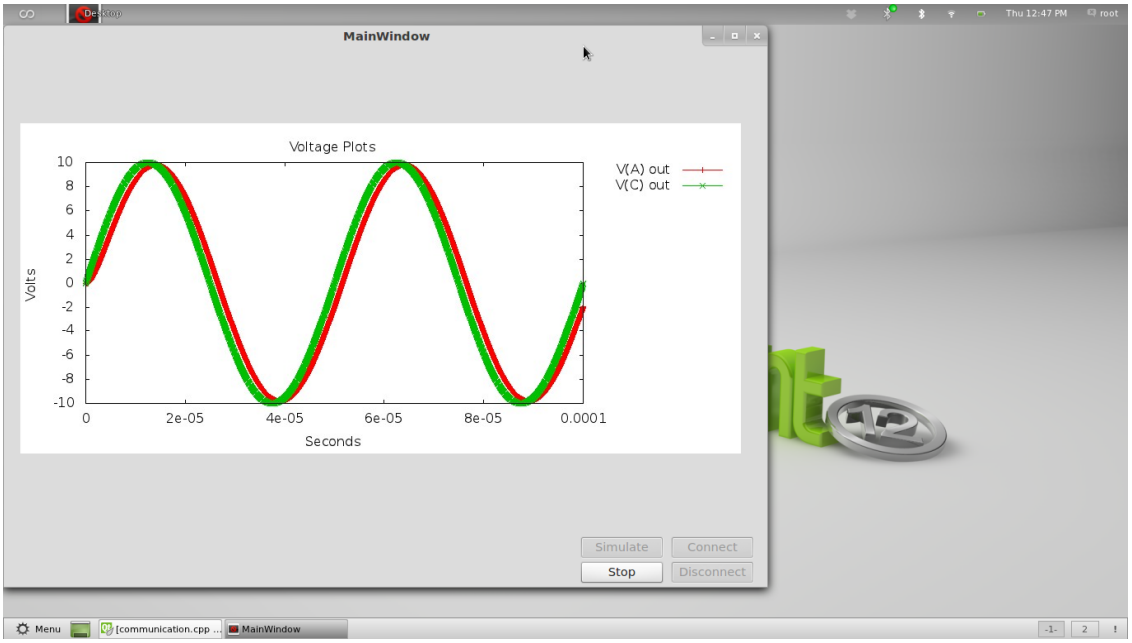


Figure 27: HOST COMPUTER SIMULATING

Figure 27 shows the GUI during a simulation of a low-pass filter. The green waveform is the AC voltage source, and the red waveform shows the output voltage.

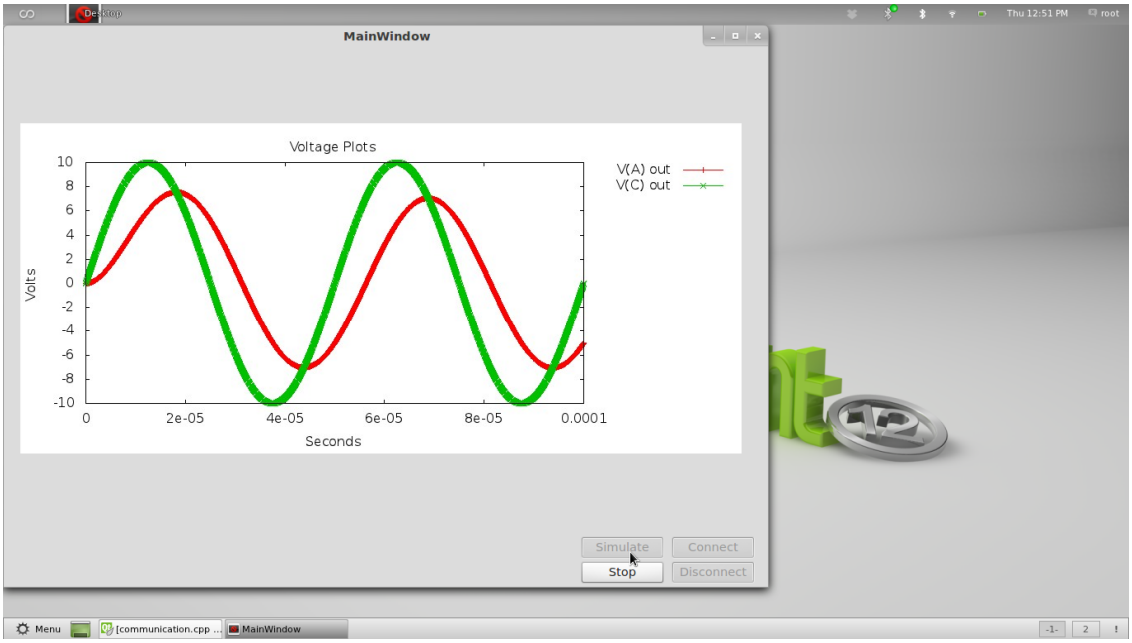


Figure 28: HOST COMPUTER UPDATE

Figure 28 shows an update to the simulation output. The resistor value in the low-pass filter was increased, resulting in a decrease in the output waveform

5.3 POSTEST

After using touchSPICE, the students participating in the survey were asked to take a posttest. Due to the number of students participating in the survey, a backup caused a wait between taking the pretest and using touchSPICE. Because of this, three students participated in the pretest, but did were unable to complete the interaction and posttest because of time constraints. The first questions asked were the exact same circuit questions from the pretest. **Table 8** shows the results of those questions on the posttest. As it can be seen, the number of correct answers increased and the number of wrong

Table 8: POSTTEST STUDENT ANSWER RESULTS

	Student's Response (number of students)		
Circuit	Right	Wrong	Don't Know
Low-pass Filter	12	4	1
Opamp	14	2	1
Parallel Resistors	15	2	0
Diode	14	3	0
Total	**55**	**11**	**2**

answers was reduced as well as the 'don't know' responses. In order to get a better grasp of what the students thought of the user interaction, additional questions were asked about their interaction with touchSPICE and their thoughts on it. Using a scale of 1-5, with 1 being the worst and 5 being the best, students were asked to rate the ease of use on touchSPICE and how easy it was to understand. Other questions asked how long it took them to simulate circuits and whether or not they would recommend touchSPICE to their friends. The results of these questions can be seen in **Table 9 – Table 11** in the next section.

VI. ANALYSIS

By using the results from the posttest, touchSPICE can be analyzed for its overall effectiveness. The first concern is results of the circuit questions. It was already shown that the number of students who got the correct answer to questions increased, but it's more important to learn where that increase in numbers came from. **Table 12** show how the results changed from the pretest to the posttest.

Table 9: CHANGES IN ANSWERS BETWEEN PRETEST AND POSTTEST

	Change in Answer				
Circuit	Don't Know to Right	Wrong to Right	Wrong to Wrong	Don't Know to Wrong	Right to Wrong
Low-pass Filter	2	4	2	3	0
Opamp	1	2	2	0	1
Parallel Resistors	0	2	0	0	2
Diode	1	4	3	0	0
Total	**4**	**12**	**7**	**3**	**3**

6.1 EFFICACY OF TOUCHSPICE AS A LEARNING TOOL

As it can be seen, there was an overall improvement in results with 16 students improving their answer, and only 13 students responding with a wrong answer. The majority of wrong answers on the posttest came from students who originally missed the same question. From watching the students use touchSPICE, a majority of those results were from students who decided to not build the tested circuits, but create their own circuits. The strange results are the ones where the student originally got a question correct, but moved to a wrong answer on the posttest. The problem related to the opamp circuit was apparent during the user study. The student who missed the opamp question

was not adjusting the feedback resistor in the circuit, but instead changed the other resistor. This action results in an inverse relation to the value of the resistor as the gain in a non-inverting opamp is: $Gain = V_{in} * \frac{R_f}{R_g}$.

The most common mistake on the diode question during the pretest was forgetting that the diode contained a voltage drop. By simulating this circuit on touchSPICE, it was easy to track the current to 20 mA, which gave a resistor value of 213.4 Ω. This shows that the voltage drop of the default diode model in SPICE has a voltage drop of 0.732V. All three students that missed this question on both pretest and posttest gave an answer of 250 Ω, which ignored the voltage drop. The graduate student who participated in the survey even went as far as to write his response with a tone showing that this question was way too easy. He wrote, "5V/x = 20mA. Using algebur in the formuler above and solving for x, we get: x = 250 ohmz."

The low-pass filter question produced mixed results for the students who originally put 'don't know' as their original response. This question shows a slight shortcoming in the touchSPICE interface. The output screen displays voltage or current in relation to time. In the low-pass filter question, the answer was related to cutoff frequency. The ideal graph to determine this answer is voltage in relation to frequency. Using touchSPICE, one method to determine the change in cutoff frequency would start off by finding a cutoff frequency. This can be done in two ways: changing the R and C values until the current frequency becomes the cutoff frequency or changing the frequency until the cutoff frequency is found. Either method is simple to do with the touchSPICE interface due to the ease of scaling values and having a real-time response

showing the output. Once the cutoff frequency has been found, a user has to change the value of the resistor and watch the output to see if it increases or decreases as the value of the resistor increases. For a student who isn't comfortable with the concept of a low-pass filter, it may be difficult to figure out how to solve this problem. Two of the three students who went from the 'don't know' response to wrong were freshmen who haven't seen this concept yet. This shows the value of a lecture to work in parallel with touchSPICE.

A majority of the users thought it was very easy to use touchSPICE. The most common comment on why it did not receive a five was the fact that it was sometimes difficult to change the values on the touchscreens during simulation. This was caused by the fast polling from the host computer. If no changes were detected, it immediately returned and asked for the information all over again. This could be improved by creating a short delay between asking for information when no simulation is required. By creating a short delay, the nodes will be able to respond faster to user inputs. Another problem was the resistive touch property. Sometimes a user would place the stylus in one spot, but the touchscreen would not read a touch in the correct zone to make a change.

Table 10: RESPONSES TO POSTTEST SURVEY QUESTIONS

	Student Response (number of students)					
Question	**1**	**2**	**3**	**4**	**5**	**Average**
How easy was it to use touchSPICE?	2	3	5	3	4	3.24
How well did you understand the results of touchSPICE?	1	2	2	10	2	3.59

Two users gave the ease of use the lowest rating of one. When reading their comments about why they gave that response, the response was all positive. One student said, “Things just made a lot of sense,” while the other student said, “Options on the touchscreen were very clear and the pins protruding on the sides of the modules made it even more clear on how to connect components together.” From these comments, it would appear that they misunderstood the points scale, believing that the lower value was a more positive result.

Similar to the response for ease of use, the responses towards understanding the results were also very positive. A few of the comments related touchSPICE to traditional SPICE simulators. One student was used to using the tradition SPICE simulator and preferred it to using touchSPICE. He commented that it was difficult to understand a low-pass filter using touchSPICE. Another student commented that touchSPICE produced a much more hands on approach compared to LTspice IV and would be a “very good” introduction to circuit analysis programs. A similar comment emphasized the benefits of wiring the circuit to know how it is connected and how text based SPICE doesn’t have this benefit.

Table 11: RESPONSES FOR RECOMMENDATIONS

	Student Response (number of students)		
Question	Yes	No	Maybe
Would you recommend touchSPICE to your friends?	16	0	1

The most commonly read comment complemented the real-time response. By being able to see changes in the graphed voltages and currents each time a small change

was made, students could really understand how their changes affected the output. Likewise, the most common negative comment was the responsiveness of the touchscreens. One student, a senior, gave poor marks in both ease (2) and understanding results (1). Looking at his responses to the questions, touchSPICE helped. This student finished the pretest with one correct answer and three incorrect answers. After taking the posttest, he improved all of these answers so that all answers were correct.

6.2 TIME TO LEARN TOUCHSPICE

Collecting user response about time required to use touchSPICE is very important. For this tool to be a valid option, it can't take an excessive amount of time to produce results. Without any instruction on using the simulator, the overall response of the students was a very low time. All of the responses except for one were under five minutes.

Table 12: TIME TO BUILD A CIRCUIT AND SIMULATE

	Student Response in minutes (number of students)						
Question	< 1	1	2	3	4	5	15
How long did it take to build and simulate a circuit using touchSPICE	5	4	1	2	1	3	1

The one outlying piece of data was 15 minutes to build and simulate a circuit. This response came from a student with Junior standing. His answer to the low-pass filter question went from don't know to wrong, but his answer to the opamp question improved from a wrong answer to a right one. To improve the time it takes to build and simulate a circuit, a tutorial session on how to use the touchSPICE tool could be implemented. There were two main things that caused students to slow down their build and simulate

time. The first hindrance was changing the component value. Most students didn't realize that the scroll bar at the bottom could be used to change the component value very quickly. The second issue involved wiring the nodes together. Even though the wires were all marked on one side to aid in the connecting of nodes, many students didn't notice this convention. Another way that students struggled was in how many connections they made. Students repeatedly connected both inputs on a node to both outputs on the neighboring node. Although this isn't going to mess up the circuit or simulation, it is unnecessary. By introducing the procedure for connecting nodes and changing values quickly, the overall time to build and simulate circuits could be improved.

6.3 REAL-TIME FEATURES

One of the key aspects to touchSPICE is its ability to simulate in real-time. The user study questions were designed to have the students build a circuit and update values to watch the response of the output. While touchSPICE was able to make updates to the simulation in real-time, the responsiveness of the touchscreens was lacking. Students interacting with the touchscreens would try to change the value of a component using the slider, but it would not respond to their stylus. The host computer was sending requests for updates from the nodes too fast. These requests caused the communication based thread on the touchscreens to occupy most of the processor time, causing the GUI thread to update slower, and at times, miss interactions.

6.4 IMPORTANCE OF THE PHYSICAL-VIRTUAL INTERFACE

The physical-virtual interface produced mixed results. Negative comments were in response to not understanding how to connect the nodes together. A simple tutorial

interfacing with touchSPICE would be an easy solution to the wiring problem. Other users appreciated the wiring interface. Users commented that using the touchscreens gave the feel of building an actual circuit compared to a standard SPICE simulator. Without this physical-virtual component, touchSPICE would be no different from every other SPICE solution that already exists. This feature combines the positives of both breadboarding circuits and SPICE simulators. Students can have the physical feel of building something, but not suffer from requiring a vast array of components and risking damage if something is built incorrectly.

6.5 SHORTCOMINGS OF ANALYSIS

The results of the user study could have been improved had the study been conducted in a different manner. With the number of students filling out the pretest, a large backup occurred and students were in a rush to complete the study. This resulted in students simulating fewer circuits than they needed to, which produced more wrong answers on the posttest than were expected. Another shortcoming resulted from not providing a tutorial on using touchSPICE. A short tutorial could have improved the users' interaction with wiring the nodes together. Improving user knowledge of building circuits would improve the time required to build circuits and increase the prevention of wiring errors, which result in failed simulations. The interfaces used in touchSPICE were not analyzed for their effectiveness. An analysis needs to be performed to test how intuitive the interfaces were, and that data could be used to improve the user experience.

VII. FUTURE WORK

With this platform, there are many possibilities to expand this project into other educational functions. The list of components supported by the SPICE simulator can be increased to encompass a wider range of circuit theories. The FriendlyARM board is currently run off of a SD Card. To change the program it's running, the new program simply needs to be loaded onto the SD Card. The operating system on the boards is not limited to Linux either. It is capable of running Android as well as WinCE. There are even different assortments of games touchSPICE can be transformed into.

Future work can be applied to the current setup of touchSPICE to improve its interface. A more responsive touchscreen could be implemented to improve user interface response time. Increasing the number of supported components would also be a relatively easy change to increase its capabilities. Subcircuits could even be implemented as a node component as long as the required inputs and outputs can be covered by the current connectors. This change would greatly increase the ability to simulate complex circuits in touchSPICE. The current method for plotting voltage and current is at a set transient response. By modifying the current host computer interface, an option for a custom transient response could be implemented.

In addition to increasing the responsiveness of the nodes, hardware changes could also improve their functionality. Reducing the form factor of the nodes will allow more nodes in a tighter space. The current design is limited to being near a power source. Developing a battery pack to power the nodes would create a much more mobile solution. Packaging the components into a custom case would increase the durability and

appearance as well. An option could be developed that would replace the wires connecting the nodes together. Utilizing the Bluetooth devices on each node, a wireless solution could be developed for discovering neighbor information, but this would reduce the physical-virtual aspect of touchSPICE.

Physical-virtual interfaces have not been a well-studied aspect in teaching circuit theory. Additional research could be done to directly compare physical, virtual and hybrid systems to evaluate their benefits as a learning tool. A human computer interface (HCI) analysis would also be beneficial to improving the touchSPICE interface both on the host computer and touchscreens. Improving the interface through the HCI study could increase the ability for students to learn using touchSPICE.

One game could involve a missing component aspect. The host computer will simulate a given circuit, but the user won't be able to see one component. In order to win the game, the user would have to change the values on the known components to figure out the specifications of the missing component. This type of game would test the user's knowledge of circuits and expected outputs to identify the part.

Another game could involve circuit building knowledge. This game would reverse the roles of the host computer and touchscreen nodes. Instead of the user telling the simulator what components need to be simulated, the host computer would send the component specifications to the individual nodes. The user would know they have completed the correct circuit by matching their output waveform to the one specified by the host computer.

VIII. CONCLUSIONS

Through the answers to the questions on the pretest and posttest, as well as the comments made by the students who participated in a limited user study, it can be shown that with an improvement to the responsiveness of the touchscreen, touchSPICE may be a viable solution for an effective method for teaching circuit theory. Results indicate that TouchSPICE improves on SPICE's shortcomings by introducing a real-time response to changes in the circuit, as well as providing a feel for creating a real circuit by exploiting physical hardware connections. Although touchSPICE produces good results in solving problems, there are still a few improvements that need to be made before it can be successfully implemented. The first improvement that needs to be made involves the response of the touchscreen. While actively simulating a circuit, the responsiveness of the touchscreen decreases. This can be corrected through a faster processor speed, or creating short delays in the polling loop on the host computer. Introducing delays introduces a tradeoff; by slowing down the polling of the nodes, the real-time aspect of the simulator is slowed down. There would be a greater delay between changes made in the circuit and seeing the results in the graph. The second improvement is a simple change that would implement a short training session for using the nodes. Overall, the touchSPICE system appears to have some merit for improving the learning experience of students on introductory-level circuit theory.

VII. BIBLIOGRAPHY

[1] Abramovitz, A.; , "Teaching Behavioral Modeling and Simulation Techniques for Power Electronics Courses," *Education, IEEE Transactions on* , vol.54, no.4, pp.523-530, Nov. 2011

[2] "Fun. Cubed." *Sifteo*. 2012. Web. <https://www.sifteo.com/product>

[3] Holtman, Marcel. "Rfcomm." *Linux Command*. 28 Apr. 2012. Web. <http://linuxcommand.org/man_pages/rfcomm1.html>.

[4] Karatza, Helen D. "A Comparative Analysis of Scheduling Policies in a Distributed System Using Simulation." *I. J. of SIMULATION* 1.1-2: 12-20.

[5] Luchetta, A.; Manetti, S.; Reatti, A.; , "SAPWIN-a symbolic simulator as a support in electrical engineering education," *Education, IEEE Transactions on* , vol.44, no.2, pp.9 pp., May 2001

[6] Lutenberg, A.; Carbonetto, S.; Inza, M.G.; Rus, D.; Venturino, G.; Natale, L.; Zacchigna, F.; , "An introductory electronics course oriented to develop real-life-engineering design skills," *Frontiers in Education Conference (FIE), 2011* , vol., no., pp.T2D-1-T2D-7, 12-15 Oct. 2011

[7] Mengmeng Zhang; Xiaohan Guan; Wenkai Liu; Changnian Zhang; , "An alternative teaching method in second-order circuit by computer-assisted instruction," *Computer Science & Education, 2009. ICCSE '09. 4th International Conference on* , vol., no., pp.1781-1784, 25-28 July 2009

[8] Pfeifer, D.; Gerstlauer, A.; , "Expression-Level Parallelism for Distributed Spice Circuit Simulation,"*Distributed Simulation and Real-time Applications (DS-RT), 2011 IEEE/ACM 15th International Symposium on* , vol., no., pp.12-17, 4-7 Sept. 2011

[9] Van Der Spiegel, Jan. "SPICE - A Brief Tutorial." *Penn Engineering*. 14 July 2010. Web. <http://www.seas.upenn.edu/~jan/spice/spice.overview.html>.

APPENDICES

A. MATERIALS

COMPONENT	PURCHASE LOCATION	QUANTITY	COST PER UNIT
Mini2440	Andahammer.com	10	$120
Mini2440 Wire Kit	Andahammer.com	10	$12
WT12-A	Inmojo.com	10	$40
Logic Level Converter	Sparkfun.com	10	$2
F/F Jumpers	IEEE-SB Lounge	Pack of 40	$8

B. NODE SCREEN SHOTS

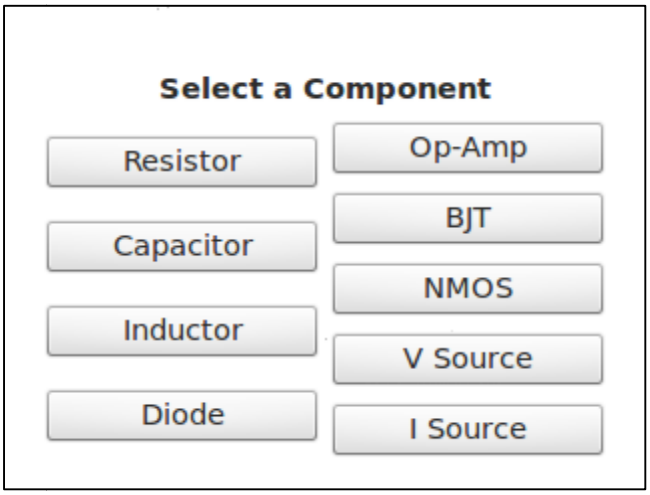


Figure 29: COMPONENT SELECTION SCREEN

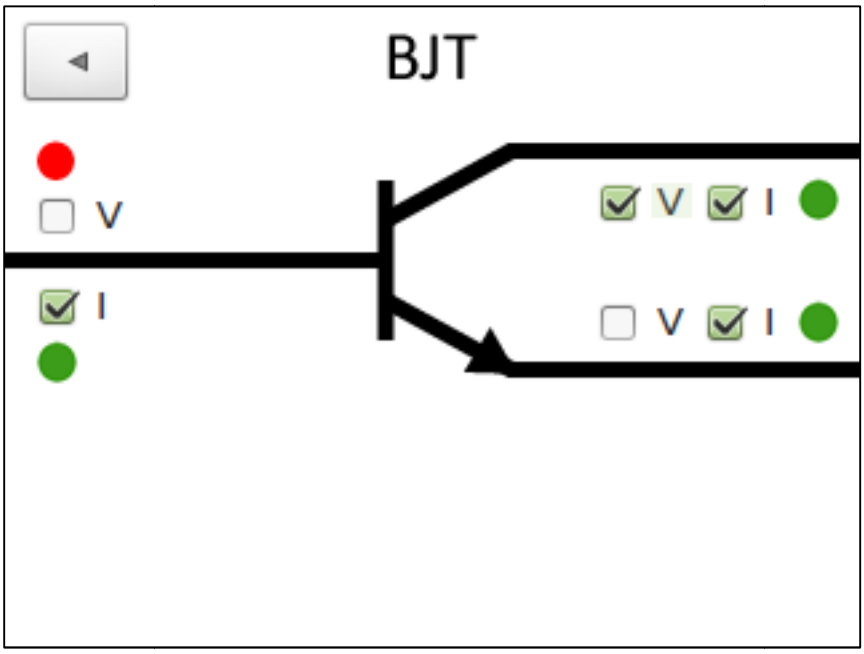


Figure 30: BJT COMPONENT SCREEN

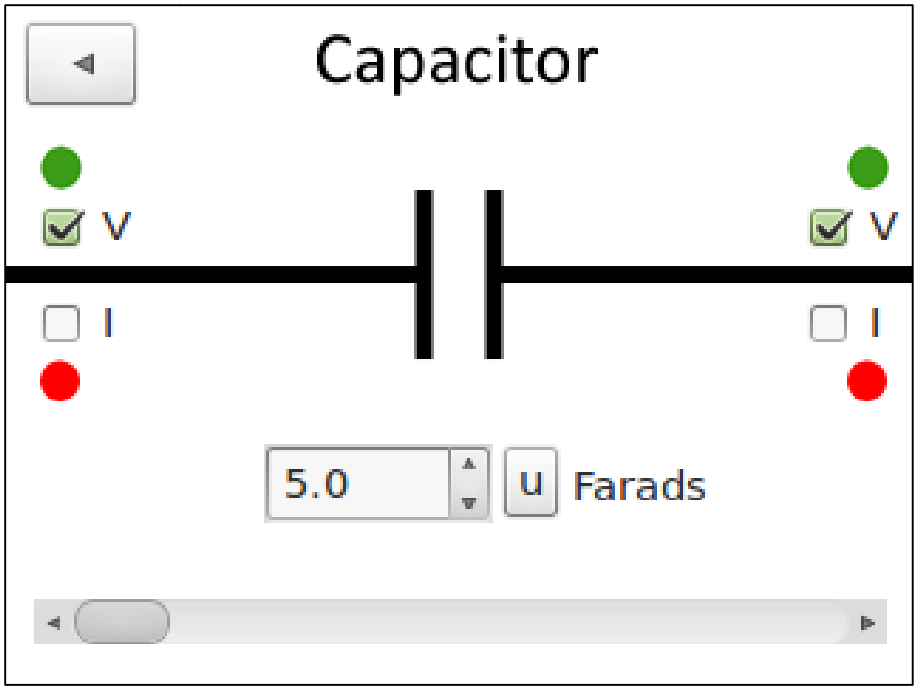


Figure 31: CAPACITOR COMPONENT SCREEN

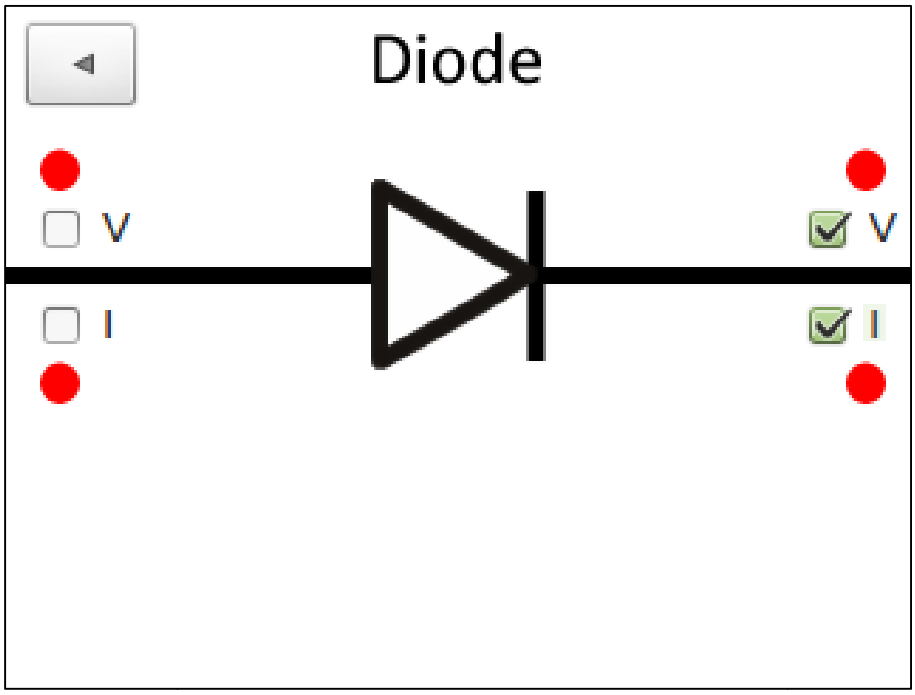


Figure 32: DIODE COMPONENT SCREEN

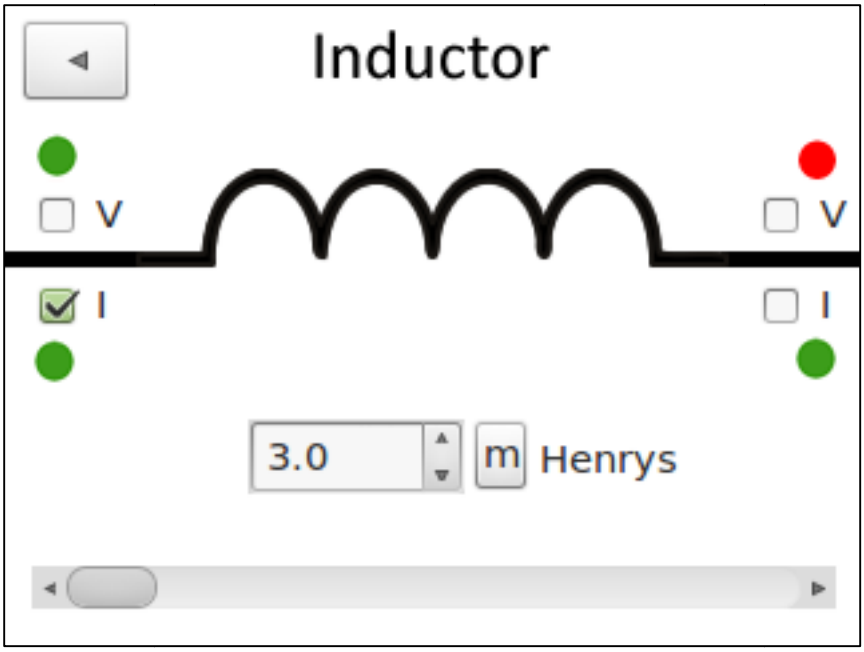


Figure 33: INDUCTOR COMPONENT SCREEN

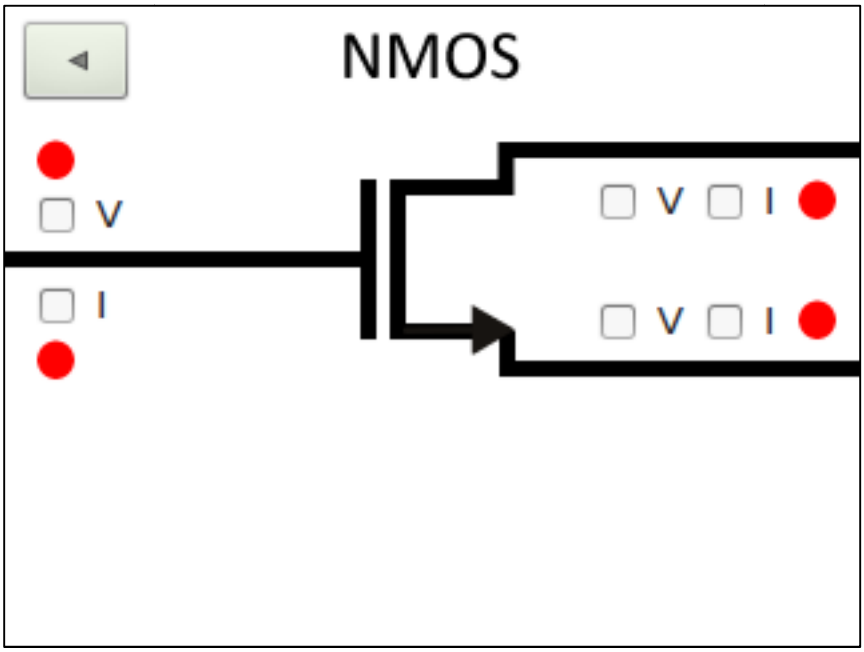


Figure 34: NMOS COMPONENT SCREEN

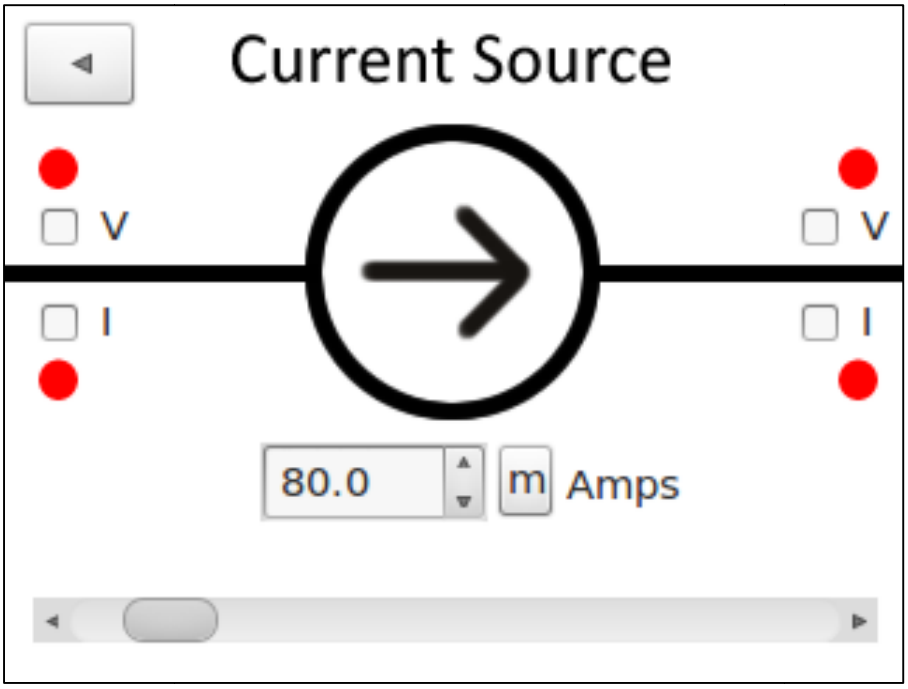


Figure 35: CURRENT SOURCE COMPONENT SCREEN

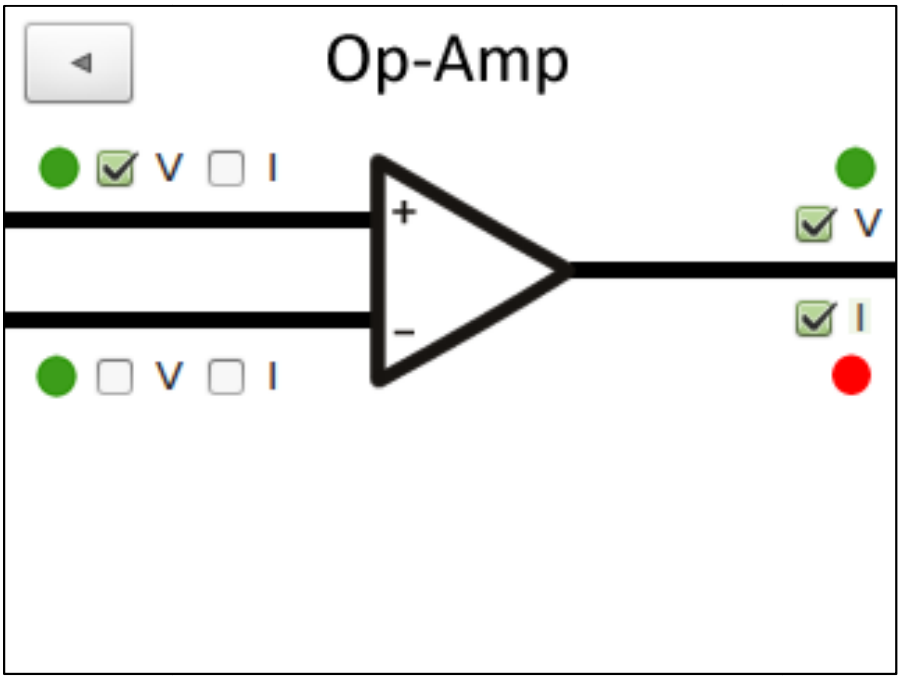


Figure 36: OPAMP COMPONENT SCREEN

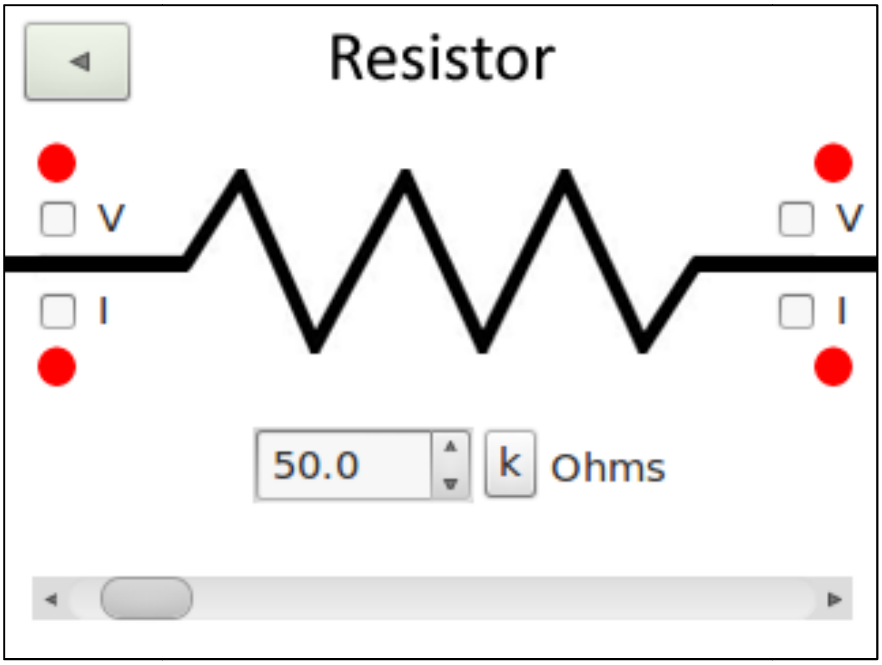


Figure 37: RESISTOR COMPONENT SCREEN

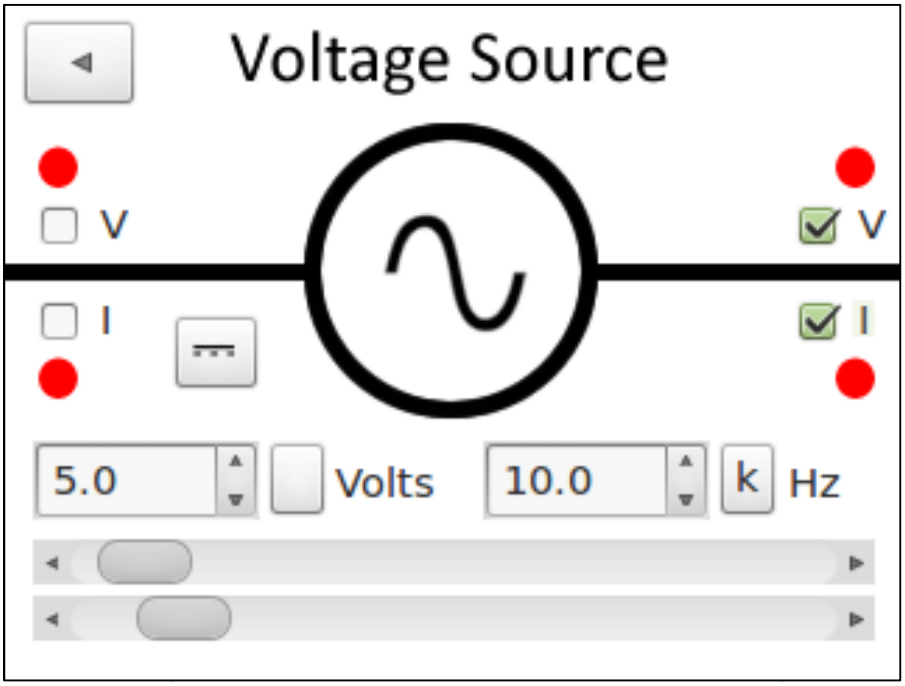


Figure 38: AC VOLTAGE SOURCE COMPONENT

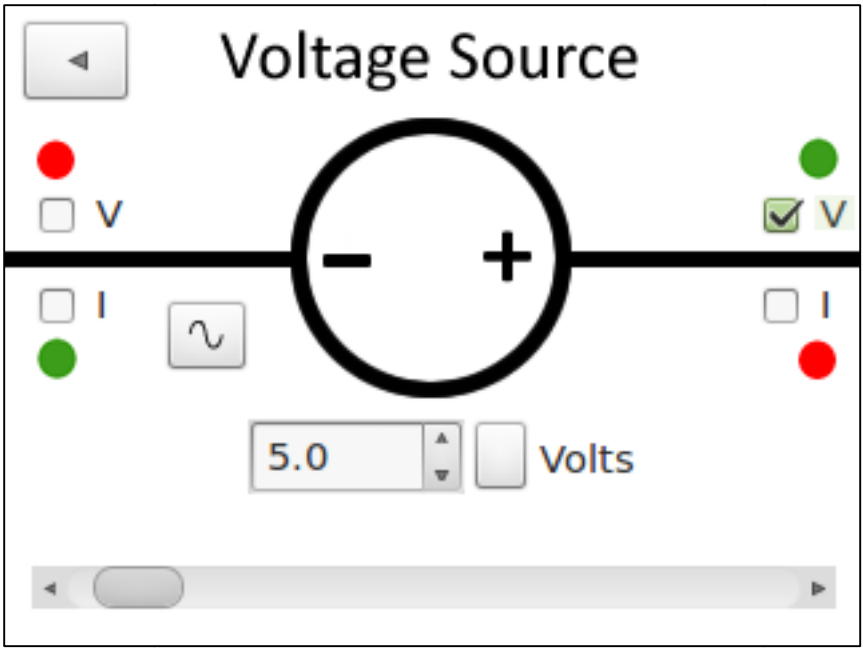


Figure 39: DC COMPONENT SOURCE

C. NODE SOFTWARE FLOW DIAGRAMS

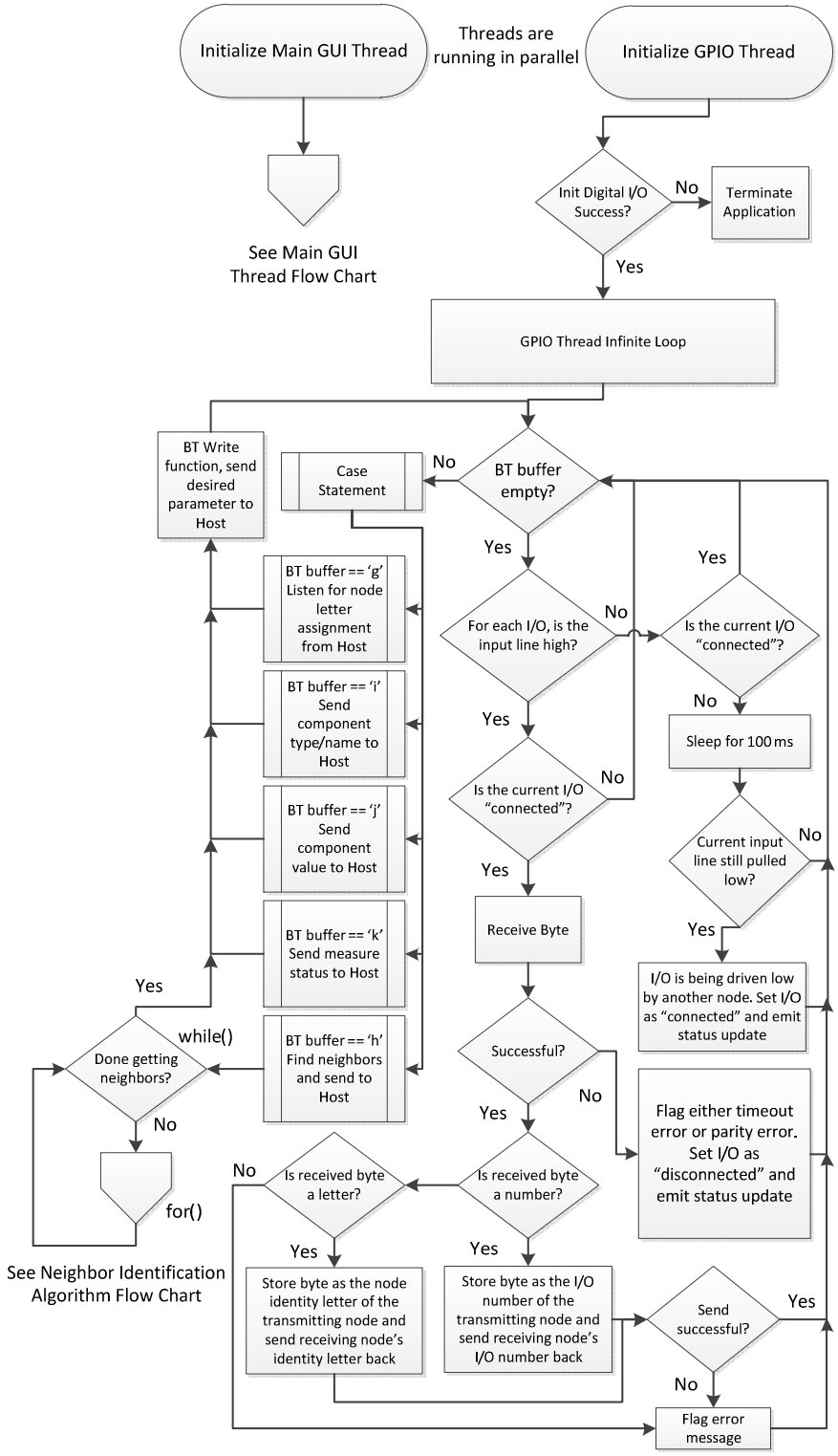


Figure 40: NODE SOFTWARE FLOW DIAGRAM

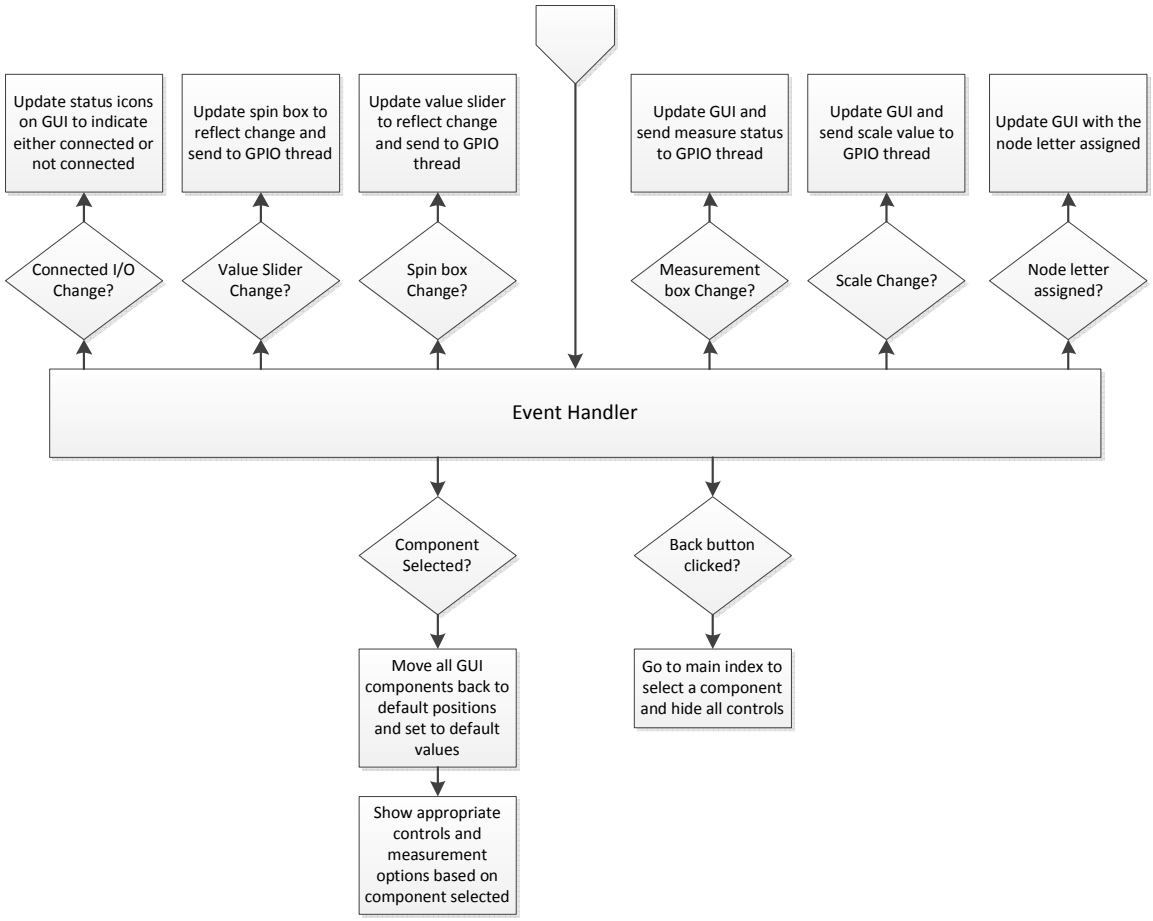


Figure 41: MAIN THREAD EVENT HANDLER

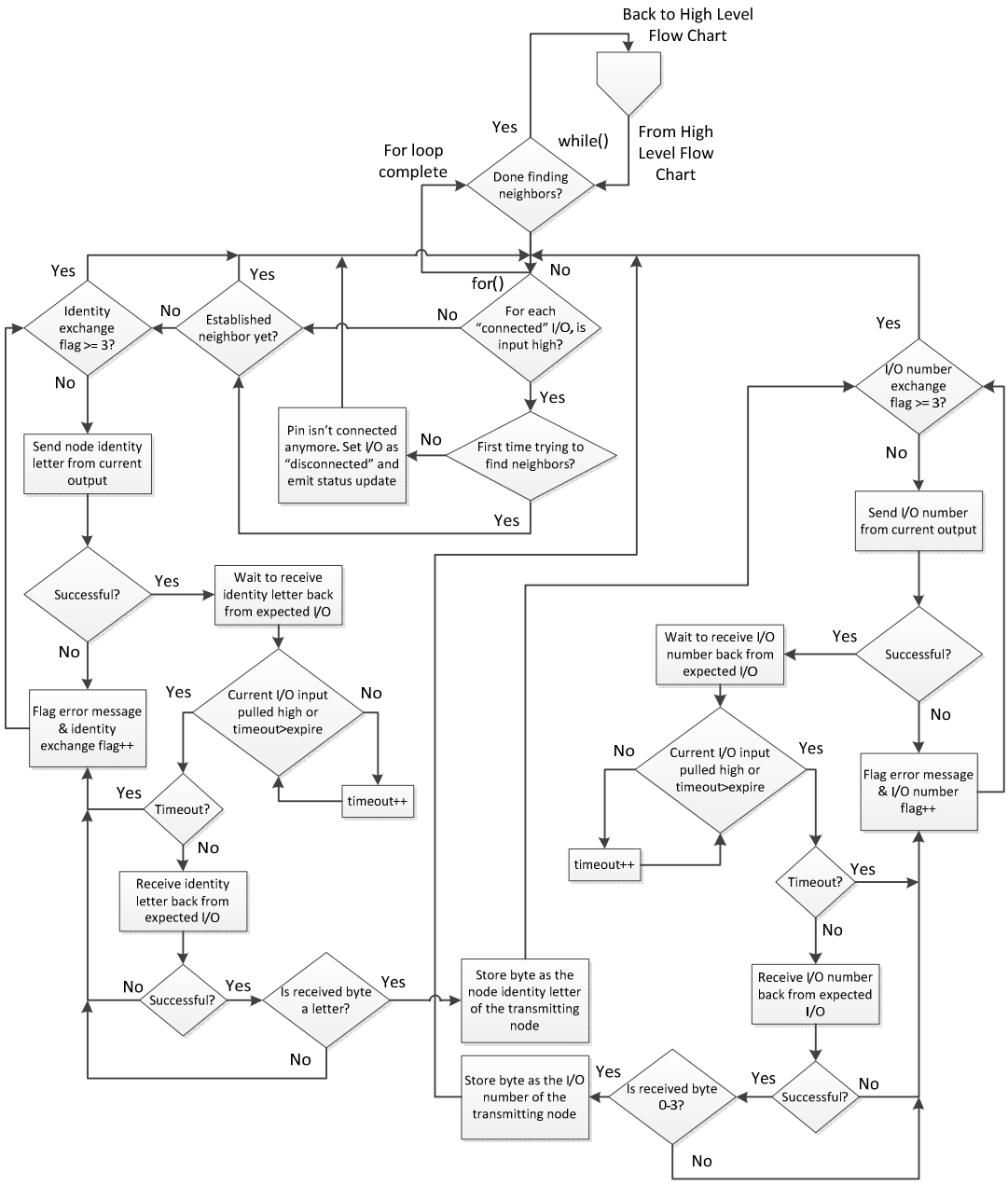


Figure 42: NODE NEIGHBOR COMMUNICATION

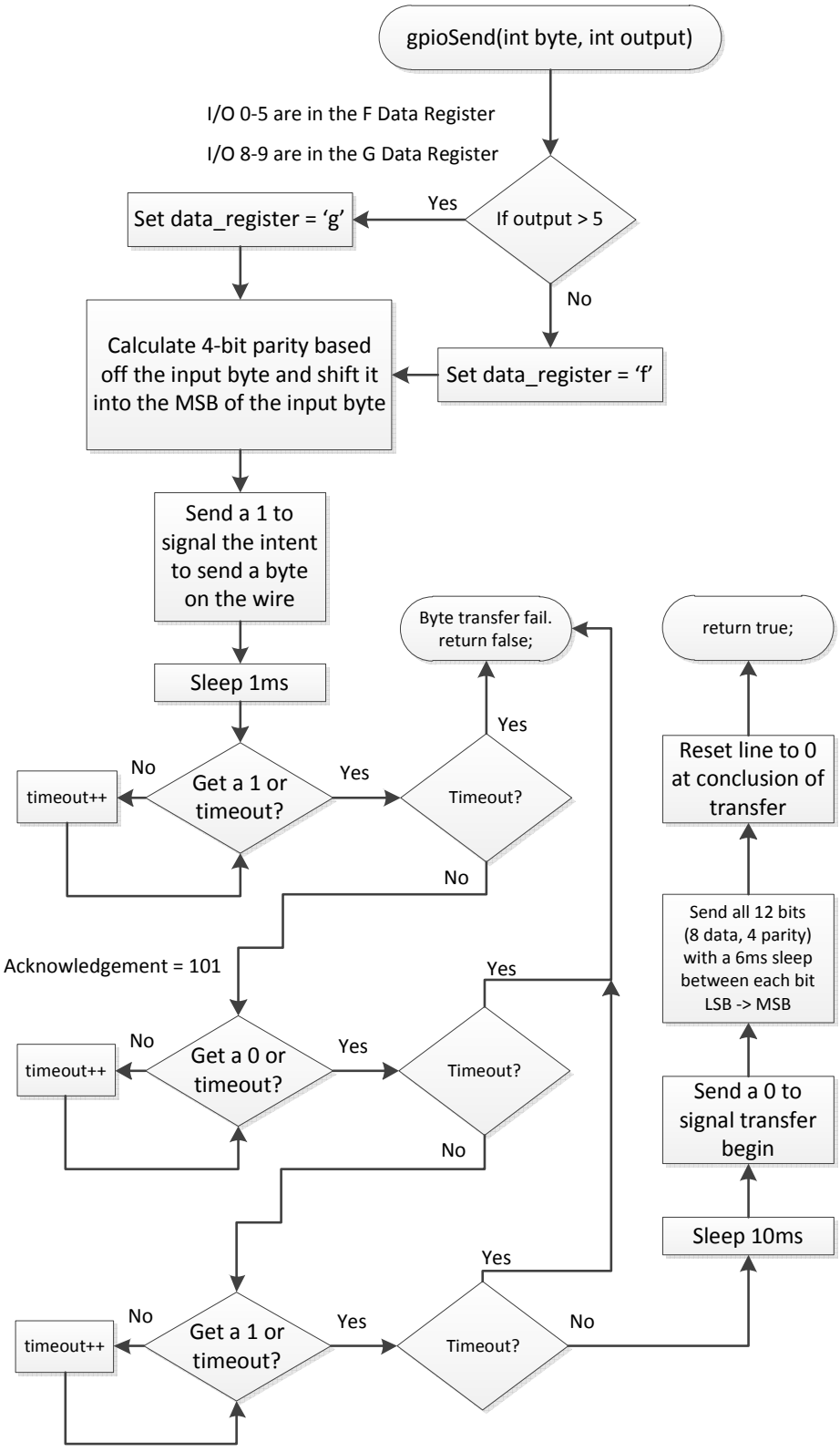


Figure 43: NODE GPIO SEND

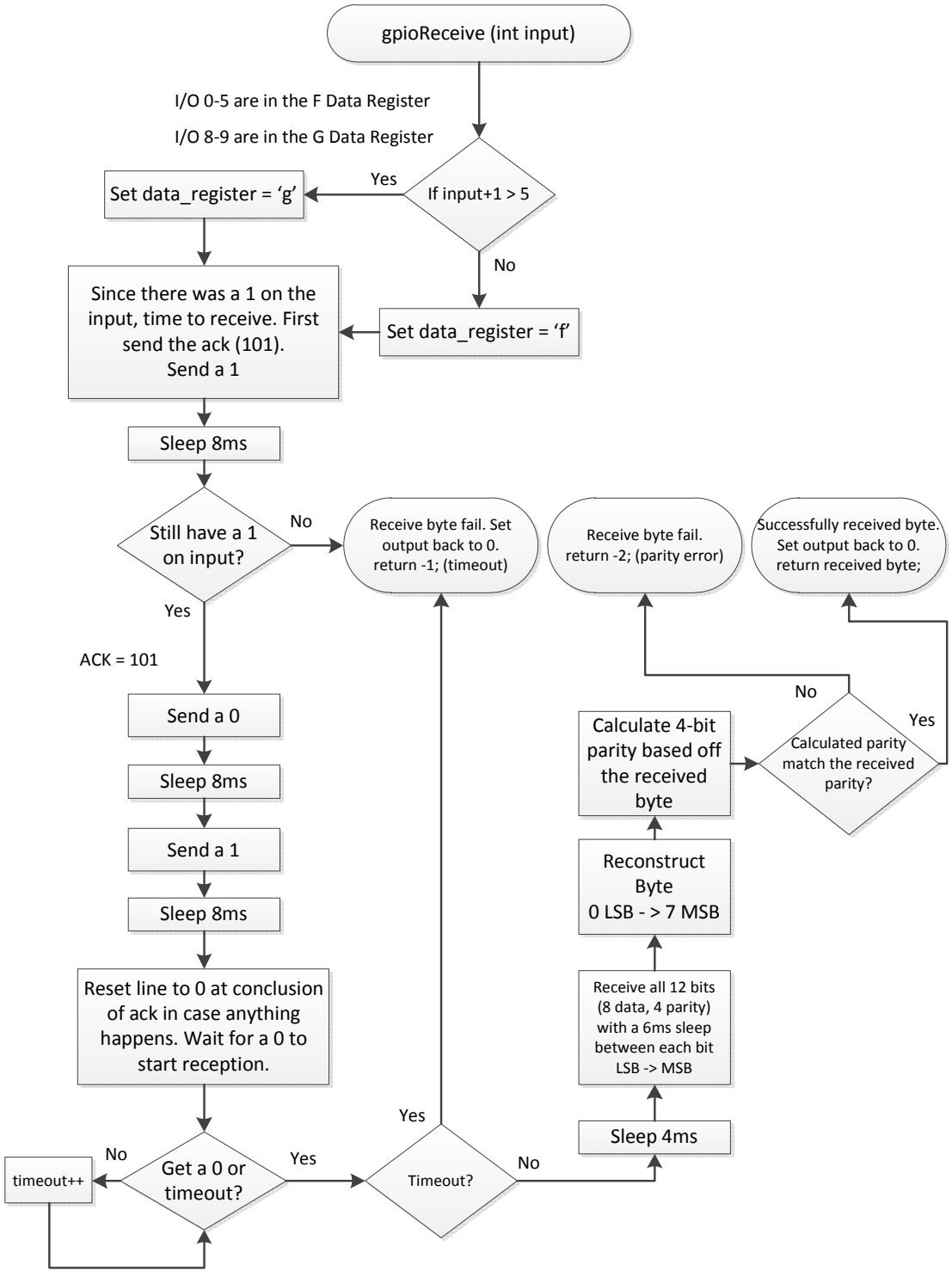


Figure 44: NODE GPIO RECEIVE

D. USER STUDY FORMS

Master's Thesis Pre-Survey

If you don't know an answer, simply write, "I don't know". Please do not make a blind guess.

* Required

User Number *We gave you this number

Grade Level *

- Freshman
- Sophomore
- Junior
- Senior
- Graduate

SPICE Simulators Used *

- LTspice IV
- HSPICE
- PSpice
- Multisim
- Micro-Cap
- None
- Other:

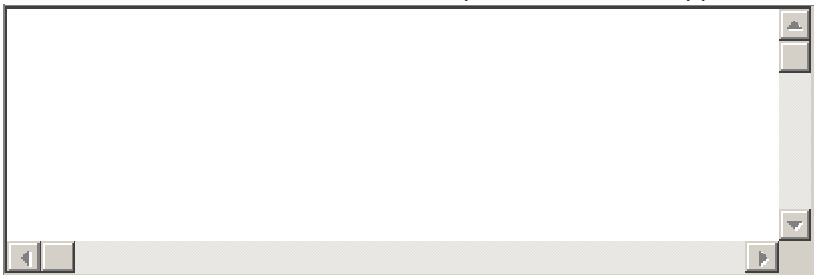
Preferred SPICE Simulator and Why

Lab Equipment Used *

- ☐ Breadboard
- ☐ Decade Resistor/Capacitor
- ☐ Oscilloscope
- ☐ Multimeter
- ☐ Function Generator
- ☐ DC Power Supply
- ☐ Sweep and Go

Other:Any other lab equipment used?

As the resistance is increased in a low-pass filter, what happens to the cutoff frequency? *



How does increasing the value of a feedback resistor on an opamp affect its gain? *

You have a circuit with a DC source and three resistors set up with the source and one resistor in series, and the other two resistors in parallel. How would decreasing the value of one of the

resistors in parallel affect current in the branches? *

With a 5V source, what value resistor is required to get 20mA of current through a diode in series? *

Master's Thesis Post-Survey

* Required

User Number *Same as before

As the resistance is increased in a low-pass filter, what happens to the cutoff frequency? *

How does increasing the value of a feedback resistor on an opamp affect its gain? *

You have a circuit with a DC source and three resistors set up with the source and one resistor in series, and the other two resistors in parallel. How would decreasing the value of one of the resistors in parallel affect current in the branches? *

With a 5V source, what value resistor is required to get 20mA of current through a diode in series? *

How well did the touchscreen nodes make it easier to understand the results? *1 is the worst, 5 is the best

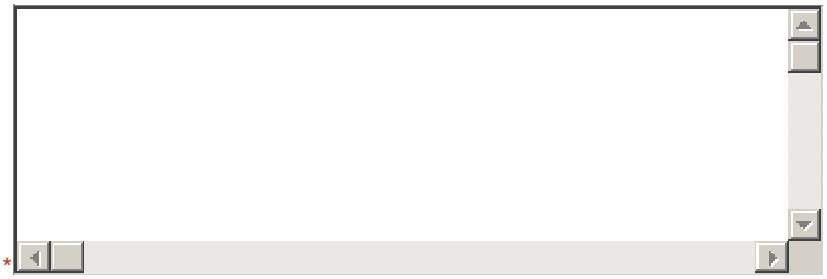
- 1
- 2
- 3
- 4
- 5

Did the touchscreen nodes make it easier to understand the results? Please Explain. *

How well did the PURE software real-time SPICE simulator make it easier to understand the results? *1 is the worst, 5 is the best

- 1
- 2
- 3
- 4
- 5

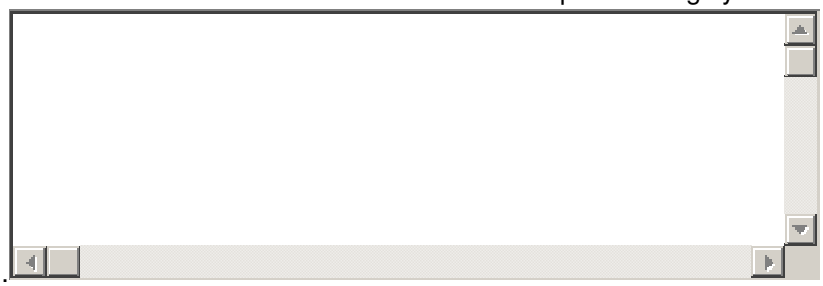
Did the PURE software real-time SPICE simulator make it easier to understand the results? Please Explain. *



How intuitive was the user interface on the touchscreens? *1 is the worst, 5 is the best

- 1
- 2
- 3
- 4
- 5

How intuitive was the user interface on the touchscreens? Explain *Things you liked and ways to improve them.



How intuitive was the user interface on the touchscreen SPICE simulator host computer? *1 is the worst, 5 is the best

- 1
- 2
- 3
- 4
- 5

How intuitive was the user interface on the touchscreen SPICE simulator host computer? Explain *Things you liked and ways to improve them.

How intuitive was the user interface on the PURE software real-time SPICE simulator? *1 is the worst, 5 is the best

- 1
- 2
- 3
- 4
- 5

How intuitive was the user interface on the PURE software real-time SPICE simulator? Explain *Things you liked and ways to improve them.

Was there anything confusing or unclear as to what needed to be done to build and simulate the circuit with the touchscreen SPICE simulator? *

Was there anything confusing or unclear as to what needed to be done to build and simulate the circuit with the PURE software real-time SPICE simulator? *

On average, how long did it take you to figure out how to choose a component, set the value, connect the wires and simulate the circuit on the touchscreen SPICE simulator? *In minutes

On average, how long did it take you to figure out how to choose a component, set the value, connect the wires and simulate the circuit on the PURE software real-time SPICE simulator? *In minutes

How easy was the new touchscreen SPICE simulator to use? *1 being easy and 5 being hard.

- 1
- 2
- 3
- 4
- 5

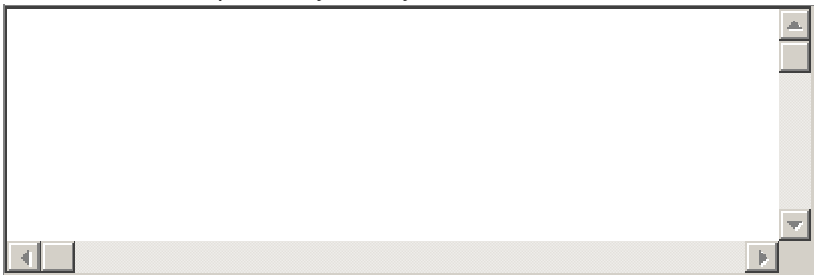
Why did you select that answer for the above question? *(touchscreen SPICE simulator)

How easy was the new PURE software real-time SPICE simulator to use? *1 being easy and 5 being hard.

- 1
- 2
- 3
- 4
- 5

Why did you select that answer for the above question? *(PURE software real-time SPICE simulator)

Would you recommend either of these real-time SPICE simulator tools to your friends as a way to learn circuit concepts? Why or why not? *



E. HOST COMPUTER SOURCE CODE

Communication.h

```
/* Kevin Peters
  * This is the header file for the communication based functions
  * used on the host computer for the touchscreen real time spice
  * simlator.
  *
  * May 2012
  */
#ifndef COMMUNICATION_H
#define COMMUNICATION_H
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#include "globals.h"
#define MAX_ATTEMPTS 3
using namespace std;
extern ofstream rfcomms_out[10];
extern ifstream rfcomms_in[10];
extern char inputNeighborA[10][5];
extern char inputNeighborB[10][5];
extern char outputNeighborA[10][5];
extern char outputNeighborB[10][5];
extern int  measureVIn[2][10];
extern int  measureVOut[2][10];
extern int  measureIIn[2][10];
extern int  measureIOut[2][10];
extern int  component[10];
extern char componentValue[10][15];
extern int  updateDetected;
void btConnect();
void btdisconnect();
void detectNeighbors();
void updateNeighbors(int);
void getPartName(int);
void getPartValue(int);
void getMeasureSettings(int);
void checkIfValueChanged(int,int);
string readInput(int);
#endif // COMMUNICATION_H
```

Communication.cpp

```
/* Kevin Peters
 * This is the file with the communication based functions
 * used on the host computer for the touchscreen real time spice
 * simlator.
 *
 * May 2012
 */
#include "communication.h"
#include <fstream>
#include <string>
using namespace std;
extern int deviceConnected[10];
extern int simulating;
// Used to open new connections to the touchscreen nodes
void btConnect()
{
    char buffer[50] = {0};
    char input[5] = {0};
    int attempts;
    // Bind rfcomms to device MAC Addresses
    system("rfcomm bind 0 00:07:80:4D:25:47"); // WT12-A
    system("rfcomm bind 1 00:07:80:4D:22:FC"); // WT12-B
    system("rfcomm bind 2 00:07:80:4D:25:48"); // WT12-C
    system("rfcomm bind 3 00:07:80:4D:22:F8"); // WT12-D
    system("rfcomm bind 4 00:07:80:4D:22:FA"); // WT12-E
    system("rfcomm bind 5 00:07:80:45:F0:7D"); // WT12-F
    system("rfcomm bind 6 00:07:80:4D:22:F6"); // WT12-G
    system("rfcomm bind 7 00:07:80:4D:22:F9"); // WT12-H
    system("rfcomm bind 8 00:07:80:45:F1:55"); // WT12-I
    system("rfcomm bind 9 00:07:80:4D:25:4D"); // WT12-J
    qDebug("Opening Connection");
    char comm_channel[15];
    // Attempt to open the output channel on each rfcomm
    for(int i=0; i < NUM_DEVICES; i++)
    {
        // Only attempt to connect devices that aren't already
        // connected
        if(!deviceConnected[i])
        {
            // Close old connections
            rfcomms_out[i].close();
            rfcomms_in[i].close();
            attempts = 0;
            sprintf(comm_channel,"/dev/rfcomm%d", i);
            qDebug(comm_channel);
            rfcomms_out[i].open(comm_channel);
            // If the connection failed to open, try again
            if(!rfcomms_out[i].is_open())
            {
                while(!rfcomms_out[i].is_open() && attempts <
                    MAX_ATTEMPTS)
                {
```

```
                    qDebug("RFCOMM OUT DID NOT OPEN FOR:
                    %s",comm_channel);
                    rfcomms_out[i].open(comm_channel);
                    attempts++;
                }
            }
        }
    }
    // Allow for Bluetooth devices to discover capabilities
    sleep(5);
    // Open input connections
    for(int i = 0; i < NUM_DEVICES; i++)
    {
        // Only attempt if the output connection succeeded
        if(rfcomms_out[i].is_open())
        {
            sprintf(comm_channel,"/dev/rfcomm%d", i);
            if(!deviceConnected[i])
            {
                rfcomms_in[i].open(comm_channel);
                // Retry if the connection fails
                if(!rfcomms_in[i].is_open())
                {
                    attempts = 0;
                    while(!rfcomms_in[i].is_open() && attempts <
                      MAX_ATTEMPTS)
                    {
                        qDebug("RFCOMM IN DID NOT OPEN FOR:
                             %s",comm_channel);
                        rfcomms_in[i].open(comm_channel);
                        attempts++;
                    }
                }
            }
            qDebug("Sending g and Char");
            sprintf(buffer,"g%c",'A' + i);
            rfcomms_out[i].write(buffer,strlen(buffer));
            rfcomms_out[i].flush();
            sprintf(input,"%s",readInput(i).c_str());
            if(input[0] != 0)
            {
                qDebug("Successful connection to node: %c", input[0]);
                deviceConnected[i] = 1;
            }
            else
            {
                qDebug("Connection not established to node: %c",'A'+i);
                deviceConnected[i] = 0;
            }
        }
        else
        {
            qDebug("Connection not established to node: %c",'A' + i);
            deviceConnected[i] = 0;
        }
    }
}
```

```
// Used to disconnect the connected nodes
void btdisconnect()
{
    char releaseCommand[20];
    // Close all output and input channels then clear bindings
    for(int i = 0; i < 10; i++)
    {
        sprintf(releaseCommand,"rfcomm release %d", i);
        rfcomms_out[i].close();
        rfcomms_in[i].close();
        system(releaseCommand);
        qDebug(releaseCommand);
        deviceConnected[i] = 0;
    }
}
// Called to go to all nodes and gather the neighbor information
void detectNeighbors()
{
    char input[255] = {0};
    for(int i=0; i < NUM_DEVICES; i++)
    {
        // Only check if the device is connected
        if(deviceConnected[i])
        {
            // h is the command to get neighbor information
            rfcomms_out[i].write("h",1);
            rfcomms_out[i].flush();
            memset(input,0,255);
            sprintf(input,"%s", (readInput(i).c_str()));
            qDebug("Input read as: %s for node %c",input,i+'A');
            // Store the returned information
            inputNeighborA[i][0] = input[0];
            inputNeighborA[i][1] = input[1];
            inputNeighborB[i][0] = input[2];
            inputNeighborB[i][1] = input[3];
            outputNeighborA[i][0] = input[4];
            outputNeighborA[i][1] = input[5];
            outputNeighborB[i][0] = input[6];
            outputNeighborB[i][1] = input[7];
        }
    }
    return;
}
// Gets the part name from the node on connection i
void getPartName(int i)
{
    char part_name[15] = {0};
    // i is the command to get part name
    rfcomms_out[i].write("i",1);
    rfcomms_out[i].flush();
    sprintf(part_name,"%s",readInput(i).c_str());
    qDebug("Partname is %s",part_name);
    // Compare the returned name with the supported components
    // and set to the value of the defined part name. Check
    // the previously stored name to see if the part changed.
    if(!strcmp(part_name, "resistor"))
    {
```

```
        checkIfValueChanged(component[i],RESISTOR);
        component[i] = RESISTOR;
        return;
    } else if(!strcmp(part_name, "capacitor"))
    {
        checkIfValueChanged(component[i],CAPACITOR);
        component[i] = CAPACITOR;
        return;
    }else if(!strcmp(part_name, "inductor"))
    {
        checkIfValueChanged(component[i],INDUCTOR);
        component[i] = INDUCTOR;
        return;
    }else if(!strcmp(part_name, "diode"))
    {
        checkIfValueChanged(component[i],DIODE);
        component[i] = DIODE;
        return;
    }else if(!strcmp(part_name, "nmos"))
    {
        checkIfValueChanged(component[i],NMOS);
        component[i] = NMOS;
        return;
    }else if(!strcmp(part_name, "bjt"))
    {
        checkIfValueChanged(component[i],BJT);
        component[i] = BJT;
        return;
    }else if(!strcmp(part_name, "opamp"))
    {
        checkIfValueChanged(component[i],OPAMP);
        component[i] = OPAMP;
        return;
    }else if(!strcmp(part_name, "vsource"))
    {
        checkIfValueChanged(component[i],VSOURCE);
        component[i] = VSOURCE;
        return;
    }else if(!strcmp(part_name, "isource"))
    {
        checkIfValueChanged(component[i],ISOURCE);
        component[i] = ISOURCE;
        return;
    }else
    {
        qDebug("NOT A COMPONENT!");
    }
}
// Gets the part value from the node on connection i
void getPartValue(int i)
{
    char part_value[25] = {0};
    // j is the command to receive part value
    rfcomms_out[i].write("j",1);
    rfcomms_out[i].flush();
    sprintf(part_value,"%s",readInput(i).c_str());
    qDebug("part value is %s",part_value);
```

```
    // Check for an update in the value
    if(strcmp(part_value,componentValue[i]))
    {
        updateDetected = 1;
    }
    sprintf(componentValue[i],"%s",part_value);
    qDebug(componentValue[i]);
}
// Gets the measurement information from the node on connection i
void getMeasureSettings(int i)
{
    char input[10] = {0};
    // Send a 'k' to initiate Measurement querry
    rfcomms_out[i].write("k",1);
    rfcomms_out[i].flush();
    sprintf(input,"%s",readInput(i).c_str());
    qDebug("Read in Measurement settings: ");
    // Convert the char to an int and check for changes
    checkIfValueChanged(measureVIn[0][i],input[0] - '0');
    checkIfValueChanged(measureVIn[1][i],input[1] - '0');
    checkIfValueChanged(measureVOut[0][i],input[2] - '0');
    checkIfValueChanged(measureVOut[1][i],input[3] - '0');
    measureVIn[0][i] = input[0] - '0';
    measureVIn[1][i] = input[1] - '0';
    measureVOut[0][i] = input[2] - '0';
    measureVOut[1][i] = input[3] - '0';
    checkIfValueChanged(measureIIn[0][i],input[4] - '0');
    checkIfValueChanged(measureIIn[1][i],input[5] - '0');
    checkIfValueChanged(measureIOut[0][i],input[6] - '0');
    checkIfValueChanged(measureIOut[1][i],input[7] - '0');
    measureIIn[0][i] = input[4] - '0';
    measureIIn[1][i] = input[5] - '0';
    measureIOut[0][i] = input[6] - '0';
    measureIOut[1][i] = input[7] - '0';
    return;
}
// Updates the neighbor information for the node on connection i
// CURRENTLY UNUSED DUE TO INSTABILITY RESULTING IN SIMULATIONS
void updateNeighbors(int i)
{
    char input[10] = {0};
    char temp[10] = {0};
    // Send 'h' to receive neighbor information
    rfcomms_out[i].write("h",1);
    rfcomms_out[i].flush();
    sprintf(input,"%s",readInput(i).c_str());
    inputNeighborA[i][0] = input[0];
    inputNeighborA[i][1] = input[1];
    inputNeighborB[i][0] = input[2];
    inputNeighborB[i][1] = input[3];
    outputNeighborA[i][0] = input[4];
    outputNeighborA[i][1] = input[5];
    outputNeighborB[i][0] = input[6];
    outputNeighborB[i][1] = input[7];
    return;
}
// Used to compare the previously stored value with the
```

```
// newly received value so updates can be detected
void checkIfValueChanged(int oldValue, int newValue)
{
    if(oldValue != newValue)
    {
        updateDetected = 1;
        qDebug("UPDATE DETECTED between %d and %d!",oldValue,newValue);
    }
    return;
}
// Reads the input buffer and returns a string of the data
string readInput(int i)
{
    char input[255] = {0};
    int j = 0;
    char temp[2] = {0};
    int timeout = 0;
    string result;
    qDebug("Attempting to read from node %c",i+'A');
    // Read the input until an '!' is received
    while(temp[0] != '!' && timeout < 100000000)
    {
        temp[0] = 0;
        rfcomms_in[i].readsome(temp,1);
        if(temp[0] && temp[0] != '!')
        {
            qDebug("%c",temp[0]);
            input[j] = temp[0];
            j++;
            timeout = 0;
        }
        if(temp[0]=='!')
        {
            qDebug("%c",temp[0]);
        }
        timeout++;
    }
    if(timeout == 100000000)
    {
        qDebug("Read timed out");
    }
    qDebug("Returning %s",input);
    result = input;
    return result;
}
```

Globals.h

```
/* Kevin Peters
 * This is the header file for the globally defined variables
 * used on the host computer for the touchscreen real time spice
 * simlator.
 *
 * May 2012
 */
#ifndef GLOBALS_H
#define GLOBALS_H
#include <QDebug>
#define NUM_DEVICES 7
#define NONE        -1
#define RESISTOR    0
#define CAPACITOR   1
#define INDUCTOR    2
#define DIODE       3
#define NMOS        4
#define BJT         5
#define OPAMP       6
#define VSOURCE     7
#define ISOURCE     8
#endif // GLOBALS_H
```

Main.cpp

```
#include <QtGui/QApplication>
#include "mainwindow.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Mainwindow.h

```
/* Kevin Peters
 * This is the header file for the main window functions
 * used on the host computer for the touchscreen real time spice
 * simlator.
 *
 * May 2012
 */
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include "communication.h"
#include "simulate_and_plot.h"
#include "globals.h"
namespace Ui {
class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_btConnect_clicked();
    void on_btDisconnect_clicked();
    void on_simulate_clicked();
    void on_stopSim_clicked();
private:
    Ui::MainWindow *ui;
    void updateSimulation();
    void toggleEnabled(bool);
};
#endif // MAINWINDOW_H
```

Mainwindow.cpp

```
/* Kevin Peters
 * This is the file with the functions related to the user interface
 * used on the host computer for the touchscreen real time spice
 * simlator.
 *
 * May 2012
 */
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <fcntl.h>
#include <iostream>
#include <fstream>
using namespace std;
// Define global variables used in all files
ofstream rfcomms_out[10];
ifstream rfcomms_in[10];
int deviceConnected[10] = {0};
int component[10];
char componentValue[10][15] = {{0}};
char inputNeighborA[10][5];
char inputNeighborB[10][5];
char outputNeighborA[10][5];
char outputNeighborB[10][5];
int measureVIn[2][10] = {{0}};
int measureVOut[2][10] = {{0}};
int measureIIn[2][10] = {{0}};
int measureIOut[2][10] = {{0}};
int updateDetected = 0;
int nodeVoltageMeasured[10] = {0};
int nodeCurrentMeasured[10] = {0};
int numOfVMeasured;
int numOfIMeasured;
int maxNodeValue = 0;
int dummyInVoltageNodeValue[10] = {0};
int maxInDummyNodeCount = 0;
int dummyOutVoltageNodeValue[10] = {0};
int maxOutDummyNodeCount = 0;
int inputNode[2][10];
int outputNode[2][10];
int simulating = 1;
// Constructor
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    btdisconnect();
}
```

```
// Exit function
MainWindow::~MainWindow()
{
    // Disconnect all nodes before closing
    btdisconnect();
    delete ui;
}
// When Connect is clicked, call the connect function
void MainWindow::on_btConnect_clicked()
{
    btConnect();
}
// When Disconnect is clicked, call the disconnect function
void MainWindow::on_btDisconnect_clicked()
{
    btdisconnect();
}
// When Simulate is clicked run the simulation loop
void MainWindow::on_simulate_clicked()
{
    // Delete the old plot to prevent displaying an old image
    remove("/root/plot.jpg");
    // Disable all buttons except 'stop'
    toggleEnabled(false);
    // Enable simulating and initial update
    simulating = 1;
    updateDetected = 1;
    qDebug("Go for Neighbor Detection");
    // Detect neighbor information
    detectNeighbors();
    qDebug("Go for Update Loop");
    // Enter simulation loop
    updateSimulation();
    return;
}
// Used to update the simulation information until stopped
void MainWindow::updateSimulation()
{
    while(simulating)
    {
        for(int i = 0; i < NUM_DEVICES;i++)
        {
            if(deviceConnected[i])
            {
                qDebug("Go for Part Name");
                // Get part name
                getPartName(i);
                // If a legitimate component...
                if(component[i] != NONE)
                {
                    qDebug("Go for Part value");
                    // Get part value
                    getPartValue(i);
                    qDebug("Go for measures");
                    // Get measure settings
                    getMeasureSettings(i);
                    qDebug("Go for updated neighbors");
```

```
                    // UNUSED DUE TO INSTABILITY
                    //updateNeighbors(i);
                }
            }
        }
        // Check to see if 'stop' has been clicked before simulating
        QCoreApplication::processEvents();
        // Exit if stopped
        if(!simulating)
        {
            break;
        }
        // Only simulate when a measurement option is selected,
        // and update detected
        if(updateDetected && measureBoxChecked())
        {
            qDebug("Writing Netlist");
            // Generate the netlist
            formatNetlist();
            qDebug("Run Spice");
            // Execute a SPICE simulation in batch mode
            system("ngspice -b -o /root/results.txt
                   /root/netlist.cir");
            qDebug("format the output file");
            // Format the output so it can be used by gnuplot
            formatSpiceOutput();
            qDebug("GNUPlot stuff");
            // Plot the results
            gnuPlot();
            // Reset the update flag
            updateDetected = 0;
        }
        // Check to see if stop has been clicked
        QCoreApplication::processEvents();
        // Display the image
        ui->label->setPixmap(QPixmap("/root/plot.jpg"));
    }
    return;
}
// When 'stop' is clicked
void MainWindow::on_stopSim_clicked()
{
    // Set the flag to leave the infinite loop
    // and enable all buttons except 'stop'
    simulating = 0;
    toggleEnabled(true);
    qDebug("STOPPING SIMULATION!");
}
// Function that toggles enabled and disabled buttons
void MainWindow::toggleEnabled(bool logic)
{
    qDebug("Enabling/Disabling GUI buttons");
    ui->stopSim->setEnabled(!logic);
    ui->simulate->setEnabled(logic);
    ui->btConnect->setEnabled(logic);
    ui->btDisconnect->setEnabled(logic);
    update();
```

```
    QCoreApplication::processEvents();
}
```

Simulate_and_plot.h

```
/* Kevin Peters
 * This is the header file for the SPICE and plotting based functions
 * used on the host computer for the touchscreen real time spice
 * simlator.
 *
 * May 2012
 */
#ifndef SIMULATE_AND_PLOT_H
#define SIMULATE_AND_PLOT_H
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#include "globals.h"
using namespace std;
extern int  nodeVoltageMeasured[10];
extern int  nodeCurrentMeasured[10];
extern int  numOfVMeasured;
extern int  numOfIMeasured;
extern int  maxNodeValue;
extern int  dummyInVoltageNodeValue[10];
extern int  maxInDummyNodeCount;
extern int  dummyOutVoltageNodeValue[10];
extern int  maxOutDummyNodeCount;
extern int  inputNode[2][10];
extern int  outputNode[2][10];
extern char inputNeighborA[10][5];
extern char inputNeighborB[10][5];
extern char outputNeighborA[10][5];
extern char outputNeighborB[10][5];
extern int  measureVIn[2][10];
extern int  measureVOut[2][10];
extern int  measureIIn[2][10];
extern int  measureIOut[2][10];
extern int  component[10];
extern char componentValue[10][15];
extern int  deviceConnected[10];
void formatSpiceOutput();
void gnuPlot();
string determineFile(int);
int voltageAndCurrentPlots();
void formatNetlist();
int tiedOutput(int);
int tiedInput(int);
void assignNodeValues();
void buildWires(char *, int);
int hasNeighbors(int);
int nodeVoltageAlreadyMeasured(int);
```

```
int nodeCurrentAlreadyMeasured(int);
void adjustForCurrentMeasurements();
int measureBoxChecked();
#endif // SIMULATE_AND_PLOT_H
```

Simulate_and_plot.cpp

```
/* Kevin Peters
 * This is the file with the functions related to the simulating
 * and plotting SPICE netlists used on the host computer for the
 * touchscreen real time spice simlator.
 *
 * May 2012
 */
#include "simulate_and_plot.h"
#include <fstream>
#include <string>
// Used to create the wire numbers used in a spice netlist
void assignNodeValues()
{
    int i,j = 0;
    char temp[5] = {0};
    int nodeCount = 1;
    // Initiate the node values to -1
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES; i++)
        {
            inputNode[j][i] = -1;
            outputNode[j][i] = -1;
        }
    }
    // Create Node 0 to start numbering all other devices.
    // Assign it to the first connected Voltage Source ground
    for(i = 0; i < NUM_DEVICES; i++)
    {
        if(deviceConnected[i] && component[i] == VSOURCE &&
           hasNeighbors(i))
        {
            // Establish ground
            inputNode[0][i] = 0;
            temp[0] = i + 'A';
            temp[1] = '0';
            // Build the wiremap for this value
            buildWires(temp,inputNode[0][i]);
            // Ground on sources are tied
            inputNode[1][i] = inputNode[0][i];
            temp[0] = i + 'A';
            temp[1] = '1';
            buildWires(temp,inputNode[1][i]);
            break;
        }
    }
```

```
    // Assign Output Node Values
    for(i = 0; i < NUM_DEVICES; i++)
    {
        if(deviceConnected[i] && component[i] != NONE &&
           hasNeighbors(i))
        {
            for(j = 0; j < 2; j++)
            {
                // Check to see if the node is unassigned
                if(outputNode[j][i] == -1)
                {
                    // Assign the next value and build wire map
                    outputNode[j][i] = nodeCount;
                    temp[0] = i + 'A';
                    temp[1] = '2'+j;
                    buildWires(temp,outputNode[j][i]);
                    // If the outputs are tied, assign and build wire
                    if(tiedOutput(i))
                    {
                        qDebug("Outputs are tied for node %c", 'A'+i);
                        outputNode[j+1][i] = outputNode[j][i];
                        temp[0] = i + 'A';
                        temp[1] = '3';
                        buildWires(temp,outputNode[j+1][i]);
                        nodeCount++;
                        break;
                    }
                    nodeCount++;
                }
                else
                {
                    // If already assigned, build wire map to value
                    temp[0] = i + 'A';
                    temp[1] = '2'+j;
                    buildWires(temp,outputNode[j][i]);
                    if(tiedOutput(i))
                    {
                        temp[0] = i + 'A';
                        temp[1] = '3';
                        buildWires(temp,outputNode[j+1][i]);
                        break;
                    }
                }
            }
        }
    }
    // Assign Input Node Values, same as above
    for(i = 0; i < NUM_DEVICES; i++)
    {
        if(deviceConnected[i] && component[i] != NONE &&
           hasNeighbors(i))
        {
            for(j = 0; j < 2; j++)
            {
                if(inputNode[j][i] == -1)
                {
                    inputNode[j][i] = nodeCount;
```

```
                    temp[0] = i + 'A';
                    temp[1] = '0'+j;
                    buildWires(temp,inputNode[j][i]);
                    if(tiedInput(i))
                    {
                        inputNode[j+1][i] = inputNode[j][i];
                        temp[0] = i + 'A';
                        temp[1] = '1';
                        buildWires(temp,inputNode[j+1][i]);
                        nodeCount++;
                        break;
                    }
                    nodeCount++;
                }
                else
                {
                    temp[0] = i + 'A';
                    temp[1] = '0' + j;
                    buildWires(temp,inputNode[j][i]);
                    if(tiedInput(i))
                    {
                        temp[0] = i + 'A';
                        temp[1] = '1';
                        buildWires(temp,inputNode[j+1][i]);
                        break;
                    }
                }
            }
        }
    }
    maxNodeValue = nodeCount;
    // Create spacing in wires for dummy nodes used in current
    // measurements
    adjustForCurrentMeasurements();
}
// Checks all nodes to see if they match the
// neighbor code that just had its value assigned
void buildWires(char * match,int nodeValue)
{
    for(int i = 0; i < NUM_DEVICES; i++)
    {
        // Compare neighbor information to node connection
        // being set. Assign other connection if tied
        if(!strcmp(inputNeighborA[i],match))
        {
            inputNode[0][i] = nodeValue;
            if(tiedInput(i))
            {
                inputNode[1][i] = nodeValue;
            }
        }
        if(!strcmp(inputNeighborB[i],match))
        {
            inputNode[1][i] = nodeValue;
            if(tiedInput(i))
            {
                inputNode[0][i] = nodeValue;
```

```
            }
        }
        if(!strcmp(outputNeighborA[i],match))
        {
            outputNode[0][i] = nodeValue;
            if(tiedOutput(i))
            {
                outputNode[1][i] = nodeValue;
            }
        }
        if(!strcmp(outputNeighborB[i],match))
        {
            outputNode[1][i] = nodeValue;
            if(tiedOutput(i))
            {
                outputNode[0][i] = nodeValue;
            }
        }
    }
    return;
}
// Use the information collected to build the netlist
void formatNetlist()
{
    // delete the old file
    remove("/root/netlist.cir");
    ofstream spiceFile("/root/netlist.cir");
    char buffer [50] = {0};
    assignNodeValues();
    int opamp = 0;
    int i, j;
    int dummyInCount = 0;
    int dummyOutCount = 0;
    memset(nodeVoltageMeasured,0,10);
    memset(nodeCurrentMeasured,0,10);
    sprintf(buffer,"*Generated Netlist\n\n");
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
    for(i = 0; i < NUM_DEVICES; i++)
    {
        memset(buffer, 0, 50);
        if(deviceConnected[i] && component[i] != NONE &&
           hasNeighbors(i))
        {
            // Determine component and insert its netlist format
            switch(component[i])
            {
            case 0:
                sprintf(buffer,"R%d %d %d %s\n", i, inputNode[0][i],
                        outputNode[0][i], componentValue[i]);
                qDebug("Component value sent to netlist
                        %s",componentValue[i]);
                break;
            case 1:
                sprintf(buffer,"C%d %d %d %s\n", i, inputNode[0][i],
                        outputNode[0][i], componentValue[i]);
                break;
```

```
            case 2:
                sprintf(buffer,"L%d %d %d %s\n", i, inputNode[0][i],
                        outputNode[0][i], componentValue[i]);
                break;
            case 3:
                sprintf(buffer,"D%d %d %d \n", i, inputNode[0][i],
                        outputNode[0][i]);
                break;
            case 4:
                sprintf(buffer,"M%d %d %d %d nmos\n", i,
                        outputNode[0][i], inputNode[0][i],
                        outputNode[1][i]);
                break;
            case 5:
                sprintf(buffer,"Q%d %d %d %d npn\n", i,
                        outputNode[0][i], inputNode[0][i],
                        outputNode[1][i]);
                break;
            case 6:
                sprintf(buffer,"X%d %d %d %d opamp741\n", i,
                        inputNode[0][i], inputNode[1][i],
                        outputNode[1][i]);
                opamp = 1;
                break;
            case 7:
                // If true, it is an AC source, else it is DC
                if(strlen(componentValue[i]) > 6)
                {
                    sprintf(buffer,"V%d %d %d sin(0 %s 0 0 0)\n", i,
                            outputNode[0][i],
                            inputNode[0][i],componentValue[i]);
                }
                else
                {
                    sprintf(buffer,"V%d %d %d %s\n", i,
                             outputNode[0][i],
                             inputNode[0][i],componentValue[i]);
                }
                break;
            case 8:
                sprintf(buffer,"I%d %d %d DC %s\n", i,
                        outputNode[0][i],
                        inputNode[0][i],componentValue[i]);
                break;
            }
            spiceFile.write(buffer,strlen(buffer));
            spiceFile.flush();
        }
    }
    // Create the LM741 Subcircuit
    if(opamp)
    {
        memset(buffer,0,50);
        sprintf(buffer,".subckt opamp741 1 2 3\nrin 1 2 2meg\nrout 6 3
                75\n");
        spiceFile.write(buffer,strlen(buffer));
        spiceFile.flush();
```

```
        memset(buffer,0,50);
        sprintf(buffer,"e 4 0 1 2 100k\nrbw 4 5 0.5meg\ncbw 5 0
                31.85nf\n");
        spiceFile.write(buffer,strlen(buffer));
        spiceFile.flush();
        memset(buffer,0,50);
        sprintf(buffer,"eout 6 0 5 0 1\n.ends opamp741\n");
        spiceFile.write(buffer,strlen(buffer));
        spiceFile.flush();
    }
    // Build dummy voltage sources to measure current
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES; i++)
        {
            if(measureIIn[j][i] && component[i] != VSOURCE)
            {
                memset(buffer,0,50);
                sprintf(buffer,"V%c %d %d DC
                        0\n",'a'+dummyInCount+dummyOutCount,
                        dummyInVoltageNodeValue[dummyInCount],inputNode
                        [j][i]);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                dummyInCount++;
            }
            if(measureIOut[j][i] && component[i] != VSOURCE)
            {
                memset(buffer,0,50);
                sprintf(buffer,"V%c %d %d DC
                        0\n",'a'+dummyInCount+dummyOutCount,
                        outputNode[j][i],dummyOutVoltageNodeValue[dummy
                        OutCount]);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                dummyOutCount++;
            }
        }
    }
    memset(buffer,0,50);
    sprintf(buffer,".control\ntran 10n 100u\nprint ");
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
    numOfVMeasured = 0;
    numOfIMeasured = 0;
    dummyInCount = 0;
    dummyOutCount = 0;
    // Insert measurement statements, but limited to 7 total
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured +
             numOfIMeasured); i++)
        {
            if(measureVIn[j][i] &&
               !nodeVoltageAlreadyMeasured(inputNode[j][i]))
            {
                memset(buffer,0,50);
```

```
                sprintf(buffer,"V(%d) ",inputNode[j][i]);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeVoltageMeasured[numOfVMeasured] = inputNode[j][i];
                numOfVMeasured++;
            }
        }
    }
    // Insert measurement statements, but limited to 7 total
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured +
             numOfIMeasured); i++)
        {
            if(measureVOut[j][i] &&
               !nodeVoltageAlreadyMeasured(outputNode[j][i]))
            {
                memset(buffer,0,50);
                sprintf(buffer,"V(%d) ",outputNode[j][i]);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeVoltageMeasured[numOfVMeasured] = outputNode[j][i];
                numOfVMeasured++;
            }
        }
    }
    // Insert measurement statements, but limited to 7 total
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured +
            numOfIMeasured); i++)
        {
            if(measureIIn[j][i] &&
               !nodeCurrentAlreadyMeasured(inputNode[j][i]))
            {
                if(component[i] == VSOURCE)
                {
                    memset(buffer,0,50);
                    sprintf(buffer,"I(V%d) ",i);
                    spiceFile.write(buffer,strlen(buffer));
                    spiceFile.flush();
                    nodeCurrentMeasured[numOfIMeasured] =
                                inputNode[j][i];
                    numOfIMeasured++;
                }
                else
                {
                    memset(buffer,0,50);
                    sprintf(buffer,"I(V%c)
                      ",'a'+dummyInCount+dummyOutCount);
                    spiceFile.write(buffer,strlen(buffer));
                    spiceFile.flush();
                    nodeCurrentMeasured[numOfIMeasured] =
                       inputNode[j][i];
                    numOfIMeasured++;
                    dummyInCount++;
                }
```

```
            }
        }
    }
    // Insert measurement statements, but limited to 7 total
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured +
              numOfIMeasured); i++)
        {
            if(measureIOut[j][i] &&
               !nodeCurrentAlreadyMeasured(outputNode[j][i]))
            {
                if(component[i] == VSOURCE)
                {
                    memset(buffer,0,50);
                    sprintf(buffer,"I(V%d) ",i);
                    spiceFile.write(buffer,strlen(buffer));
                    spiceFile.flush();
                    nodeCurrentMeasured[numOfIMeasured] =
                          outputNode[j][i];
                    numOfIMeasured++;
                }
                else
                {
                    memset(buffer,0,50);
                    sprintf(buffer,"I(V%c)
                        ",'a'+dummyInCount+dummyOutCount);
                    spiceFile.write(buffer,strlen(buffer));
                    spiceFile.flush();
                    nodeCurrentMeasured[numOfIMeasured] =
                         outputNode[j][i];
                    numOfIMeasured++;
                    dummyOutCount++;
                }
            }
        }
    }
    memset(buffer,0,50);
    sprintf(buffer,"\n.endc\n.end");
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
    return;
}
// The output from spice needs to be formatted
// before gnuplot can use it.
void formatSpiceOutput()
{
    remove("/root/formattedPart1.txt");
    remove("/root/formattedPart2.txt");
    remove("/root/formattedPart3.txt");
    ifstream inFile("/root/results.txt");
    ofstream outFile1("/root/formattedPart1.txt");
    ofstream outFile2("/root/formattedPart2.txt");
    ofstream outFile3("/root/formattedPart3.txt");
    string line;
    int indexFound = 0;
    int i = -2;
```

```
    int j = 1;
    int goodLine = 0;
    char indexCount[10];
    while(inFile.good())
    {
        // Read the next line and add a newline to the end of it
        getline(inFile,line);
        line.append("\n");
        // If the word index hasn't been found yet, look for it
        if(!indexFound)
        {
            if(line.find("Index") == 0)
            {
                indexFound = 1;
            }
        }
        // If the line count is increasing, still good
        if(i >= 0)
        {
            sprintf(indexCount,"%d",i);
            if(line.find(indexCount) == 0)
            {
                goodLine = 1;
                i++;
            }
            else
            {
                // If it resets to 0, its a new measurement setting
                if(line.find("0") == 0)
                {
                    i = 1;
                    goodLine = 1;
                    j++;
                }
            }
        }
        else
        {
            if(indexFound)
            {
                i++;
            }
        }
        if (i >= 0 && goodLine)
        {
            switch(j)
            {
                case 1:
                    outFile1.write(line.c_str(),line.length());
                    break;
                case 2:
                    outFile2.write(line.c_str(),line.length());
                    break;
                case 3:
                    outFile3.write(line.c_str(),line.length());
                    break;
            }
```

```
            goodLine = 0;
        }
    }
    inFile.close();
    outFile1.close();
    outFile2.close();
    outFile3.close();
    return;
}
void gnuPlot()
{
    remove("/root/gnuscript");
    ofstream script("/root/gnuscript");
    char buffer[100];
    char initiate[10] = "plot \\\n";
    string fileName;
    int column = 3;
    int j, i, k = 0;
    int voltageCount = 0;
    int currentCount = 0;
    memset(nodeVoltageMeasured, 0, 10);
    memset(nodeCurrentMeasured, 0, 10);
    // settings to plot graphs
    sprintf(buffer,"set term jpeg size 875,400\nset output
             \"/root/plot.jpg\"\n");
    script.write(buffer,strlen(buffer));
    script.flush();
    sprintf(buffer,"set size 1,1\nset origin 0,0\nset key outside\n");
    script.write(buffer,strlen(buffer));
    script.flush();
    // Setup the plot command
    if(voltageAndCurrentPlots())
    {
        sprintf(buffer,"set y2label 'Current (A)'\nset ytics
                 nomirror\nset y2tics auto\n");
        script.write(buffer,strlen(buffer));
    }
    sprintf(buffer,"set xlabel 'Seconds'\nset ylabel 'Volts'\nset
            x2label 'Voltage Plots'\n");
    script.write(buffer,strlen(buffer));
    script.write(initiate,strlen(initiate));
    qDebug("%d",numOfVMeasured + numOfIMeasured);
    // Add all traces to plot
    // Voltage Inputs
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
        {
            if(measureVIn[j][i] &&
               !nodeVoltageAlreadyMeasured(inputNode[j][i]))
            {
                fileName = determineFile(k);
                sprintf(buffer,"\"%s\" using 2:%d with linespoints axis
                        x1y1 title 'V(%c)
                        in'",fileName.c_str(),column,i+'A');
                script.write(buffer,strlen(buffer));
                column++;
```

```
                k++;
                nodeVoltageMeasured[voltageCount] = inputNode[j][i];
                voltageCount++;
                if(voltageCount + currentCount < numOfVMeasured +
                    numOfIMeasured)
                {
                    sprintf(buffer,", \\\n");
                    script.write(buffer,strlen(buffer));
                }
            }
            if (column > 5)
            {
                column = 3;
            }
        }
    }
    // Voltage Outputs
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
        {
            if(measureVOut[j][i] &&
               !nodeVoltageAlreadyMeasured(outputNode[j][i]))
            {
                fileName = determineFile(k);
                sprintf(buffer,"\"%s\" using 2:%d with linespoints axis
                        x1y1 title 'V(%c)
                        out'",fileName.c_str(),column,i+'A');
                script.write(buffer,strlen(buffer));
                column++;
                k++;
                nodeVoltageMeasured[voltageCount] = outputNode[j][i];
                voltageCount++;
                if(voltageCount + currentCount < numOfVMeasured +
                    numOfIMeasured)
                {
                    sprintf(buffer,", \\\n");
                    script.write(buffer,strlen(buffer));
                }
            }
            if (column > 5)
            {
                column = 3;
            }
        }
    }
    // Current Inputs
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
        {
            if(measureIIn[j][i] &&
                !nodeCurrentAlreadyMeasured(inputNode[j][i]))
            {
                fileName = determineFile(k);
```

```
                sprintf(buffer,"\"%s\" using 2:%d with linespoints axis
                        x1y2 title 'I(%c)
                        in'",fileName.c_str(),column,i+'A');
                script.write(buffer,strlen(buffer));
                column++;
                k++;
                nodeCurrentMeasured[currentCount] = inputNode[j][i];
                currentCount++;
                if(voltageCount + currentCount < numOfVMeasured +
                    numOfIMeasured)
                {
                    sprintf(buffer,", \\\n");
                    script.write(buffer,strlen(buffer));
                }
            }
            if (column > 5)
            {
                column = 3;
            }
        }
    }
    // Current Outputs
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
        {
            if(measureIOut[j][i] &&
                !nodeCurrentAlreadyMeasured(outputNode[j][i]))
            {
                fileName = determineFile(k);
                sprintf(buffer,"\"%s\" using 2:%d with linespoints axis
                         x1y2 title 'I(%c)
                         out'",fileName.c_str(),column,i+'A');
                script.write(buffer,strlen(buffer));
                column++;
                k++;
                nodeCurrentMeasured[currentCount] = outputNode[j][i];
                currentCount++;
                if(voltageCount + currentCount < numOfVMeasured +
                   numOfIMeasured)
                {
                    sprintf(buffer,", \\\n");
                    script.write(buffer,strlen(buffer));
                }
            }
            if (column > 5)
            {
                column = 3;
            }
        }
    }
    script.close();
    system("gnuplot /root/gnuscript");
    return;
}
// Determine which file the measurement values are
// stored in and return the file name in a string
```

```
string determineFile(int i)
{
    string fileName = "";
    switch(i)
    {
    case 0:
    case 1:
    case 2:
        fileName = "/root/formattedPart1.txt";
        break;
    case 3:
    case 4:
    case 5:
        fileName = "/root/formattedPart2.txt";
        break;
    case 6:
    case 7:
    case 8:
        fileName = "/root/formattedPart3.txt";
        break;
    }
    return fileName;
}
// Check to see if both voltage and current is measured
int voltageAndCurrentPlots()
{
    int i, j = 0;
    int voltage, current = 0;
    for(i=0; i < 2; i++)
    {
        for(j=0;j<10;j++)
        {
            if(measureVIn[i][j] || measureVOut[i][j])
            {
                voltage = 1;
                break;
            }
        }
        if(voltage)
        {
            break;
        }
    }
    for(i=0; i < 2; i++)
    {
        for(j=0;j<10;j++)
        {
            if(measureIIn[i][j] || measureIOut[i][j])
            {
                current = 1;
                break;
            }
        }
        if(current)
        {
            break;
        }
```

```
    }
    return (current & voltage);
}
// If the outputs are tied together, return 1
int tiedOutput(int i)
{
    switch(component[i])
    {
    case 0:
    case 1:
    case 2:
    case 3:
    case 6:
    case 7:
    case 8:
        return 1;
        break;
    case 4:
    case 5:
        return 0;
        break;
    }
    return 0;
}
// If the inputs are tied together, return 1
int tiedInput(int i)
{
    switch(component[i])
    {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
    case 7:
    case 8:
        return 1;
        break;
    case 6:
        return 0;
        break;
    }
    return 0;
}
// Find out if a node is connected to other nodes (not isolated)
int hasNeighbors(int i)
{
    if(inputNeighborA[i][0] == 'N' && inputNeighborB[i][0] == 'N' &&
       outputNeighborA[i][0] == 'N' && outputNeighborB[i][0] =='N')
    {
        return 0;
    }
    else
    {
        return 1;
    }
```

```
}
// Check if a node current has already been measured, to not
// repeat measurements
int nodeCurrentAlreadyMeasured(int nodeNumber)
{
    qDebug("Node values that have been measured for Current:");
    for(int i = 0; i < 10; i++)
    {
        qDebug("%d",nodeCurrentMeasured[i]);
        if(nodeNumber == nodeCurrentMeasured[i])
        {
            return 1;
        }
    }
    return 0;
}
// Check if a node voltage has already been measured, to not
// repeat measurements
int nodeVoltageAlreadyMeasured(int nodeNumber)
{
    qDebug("Node values that have been measured for Voltage:");
    for(int i = 0; i < 10; i++)
    {
        qDebug("%d",nodeVoltageMeasured[i]);
        if(nodeNumber == nodeVoltageMeasured[i])
        {
            return 1;
        }
    }
    return 0;
}
// Adjust node values for dummy voltage nodes
void adjustForCurrentMeasurements()
{
    int i,j;
    maxInDummyNodeCount = 0;
    maxOutDummyNodeCount = 0;
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES; i++)
        {
            if(measureIIn[j][i])
            {
                if(component[i] != VSOURCE)
                {
                    dummyInVoltageNodeValue[maxInDummyNodeCount] =
                            inputNode[j][i];
                    inputNode[j][i] = maxNodeValue;
                    maxInDummyNodeCount++;
                    maxNodeValue++;
                }
            }
        }
    }
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES; i++)
```

```
        {
            if(measureIOut[j][i])
            {
                if(component[i] != VSOURCE)
                {
                    dummyOutVoltageNodeValue[maxOutDummyNodeCount] =
                        outputNode[j][i];
                    outputNode[j][i] = maxNodeValue;
                    maxOutDummyNodeCount++;
                    maxNodeValue++;
                }
            }
        }
    }
}
// Check for a measurement option
int measureBoxChecked()
{
    int i;
    for(i = 0;i < NUM_DEVICES; i++)
    {
        if(measureVIn[0][i] || measureVIn[1][i])
        {
            return 1;
        }
        if(measureVOut[0][i] || measureVOut[1][i])
        {
            return 1;
        }
        if(measureIIn[0][i] || measureIIn[1][i])
        {
            return 1;
        }
        if(measureIOut[0][i] || measureIOut[1][i])
        {
            return 1;
        }
    }
    return 0;
}
```